Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

MACHINE LEARNING

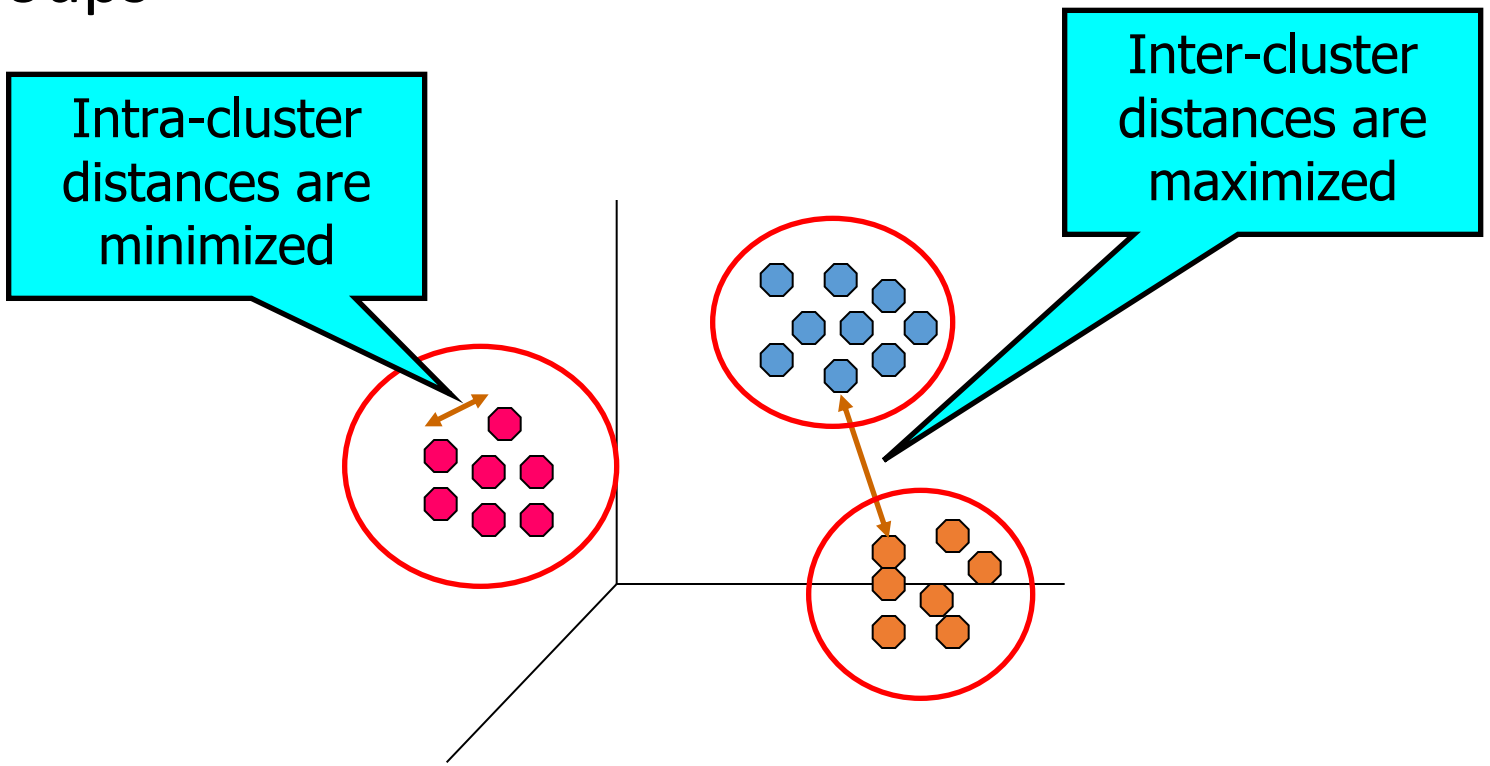
Professional Core(CET3006B)
T. Y. B.Tech CSE, Sem-VI
2023-2024
UNIT-III

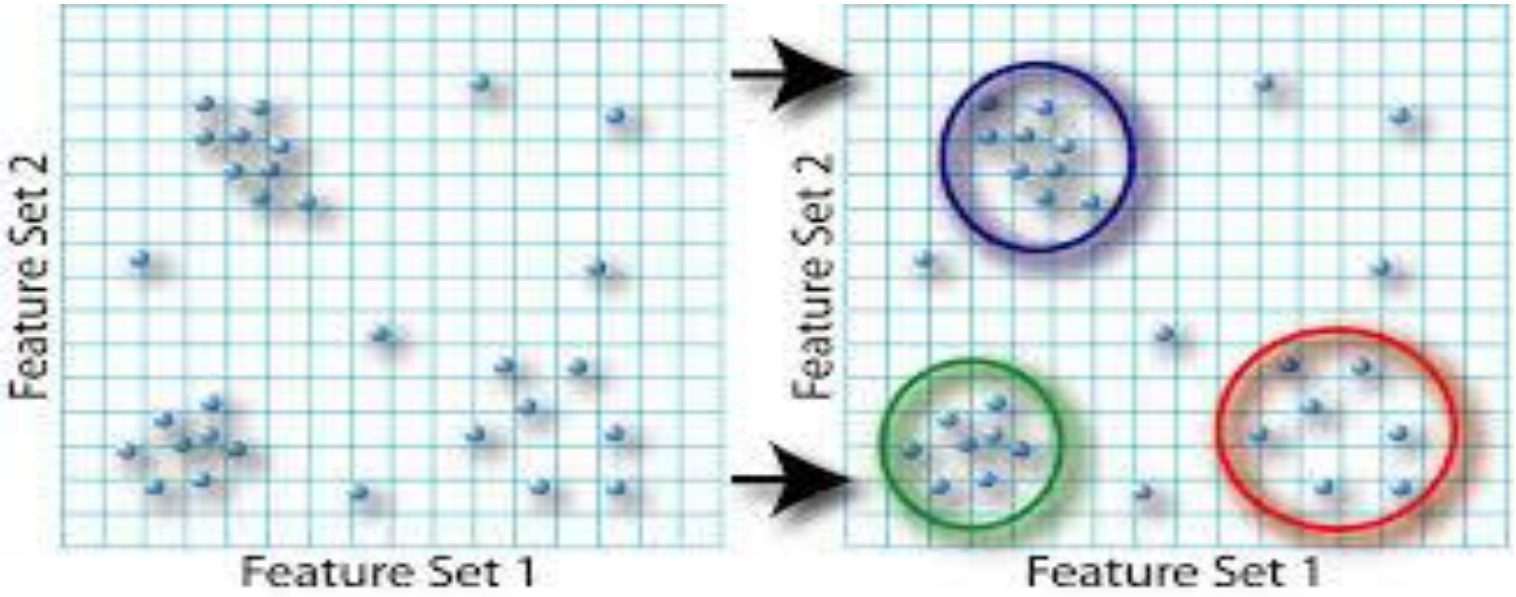
SoCSE – Dept. of Computer Engineering & Technology

- **Clustering** is the classification of objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait - often according to some defined distance measure.

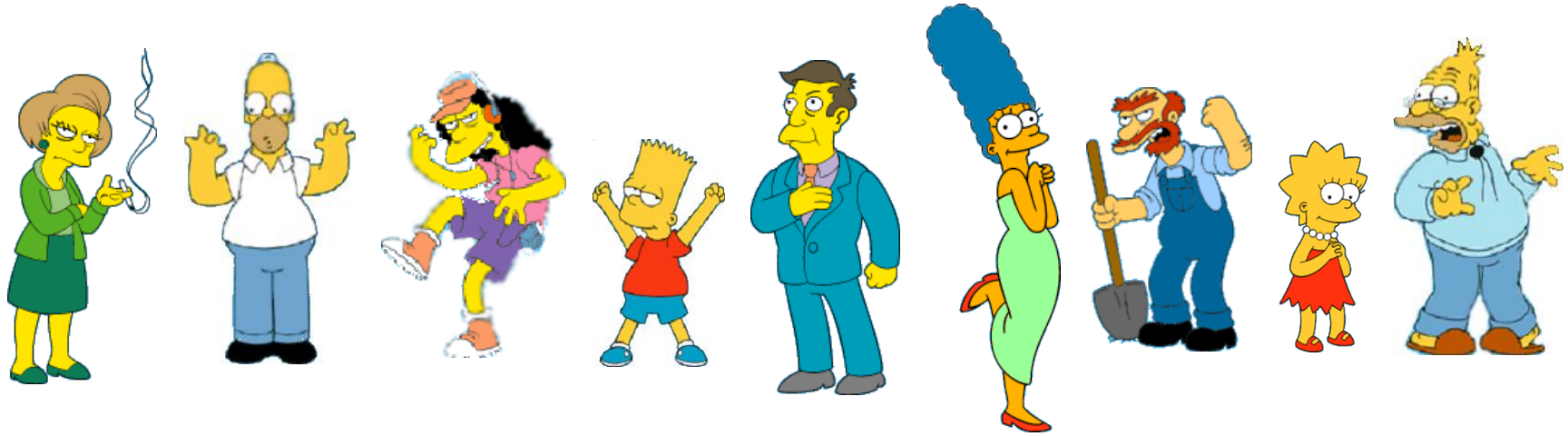
- Organizing data into classes such that there is
  - high intra-class similarity
  - low inter-class similarity
- Finding the class labels and the number of classes directly from the data (in contrast to classification).
- More informally, finding natural groupings among objects.
- Also called unsupervised learning, sometimes called classification by statisticians and sorting by psychologists and segmentation by people in marketing

- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups

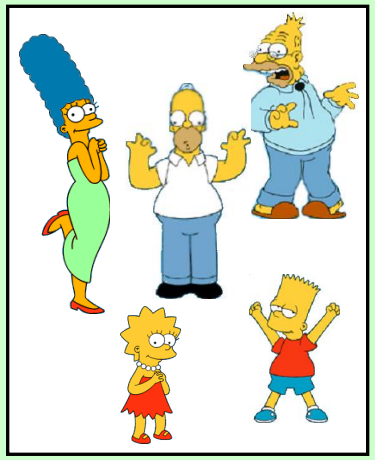



Feature Set 2
Feature Set 1
Feature Set 2
Feature Set 1

What is a natural grouping among these objects?

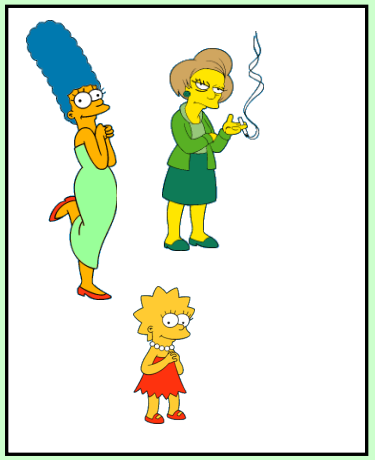


Clustering is subjective



Simpson's Family

School Employees



Females

Males

What is Similarity?

The quality or state of being similar; likeness; resemblance; as, a similarity of features.

Webster's Dictionary

Similarity is hard to define, but…
"*We know it when we see it*"

The real meaning of similarity is a philosophical question. We will take a more pragmatic approach.

Distance measure will determine how the *similarity* of two elements is calculated and it will influence the shape of the clusters.

They include:

1. The Euclidean distance (also called 2-norm distance) is given by:

$$d(x, y) = \sqrt[2]{\sum_{i=1}^{p} | x_i - y_i |^2}$$

1. The Manhattan distance (also called taxicab norm or 1-norm) is given by:

$$d(x, y) = \sum_{i=1}^{p} | x_i - y_i |$$

D(A,B) = D(B,A) *Symmetry*
Otherwise you could claim “Alex looks like Bob, but Bob looks nothing like Alex.”

D(A,A) = 0 *Constancy of Self-Similarity*
Otherwise you could claim “Alex looks more like Bob, than Bob does.”

D(A,B) = 0 IIf A=B *Positivity (Separation)*
Otherwise there are objects in your world that are different, but you cannot tell apart.

D(A,B) ≤ D(A,C) + D(B,C) *Triangular Inequality*
Otherwise you could claim “Alex is very much like Bob, and Alex is very much like Carl, but Bob is very much unlike Carl.”
Also, one can use weighted distance, and many other similarity/distance measures.

9

- Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- Land use: Identification of areas of similar land use in an earth observation database
- Insurance: Identifying groups of motor insurance policy holders with a high average claim cost
- City-planning: Identifying groups of houses according to their house type, value, and geographical location
- Earth-quake studies: Observed earth quake epicenters should be clustered along continent faults

Clustering Applications

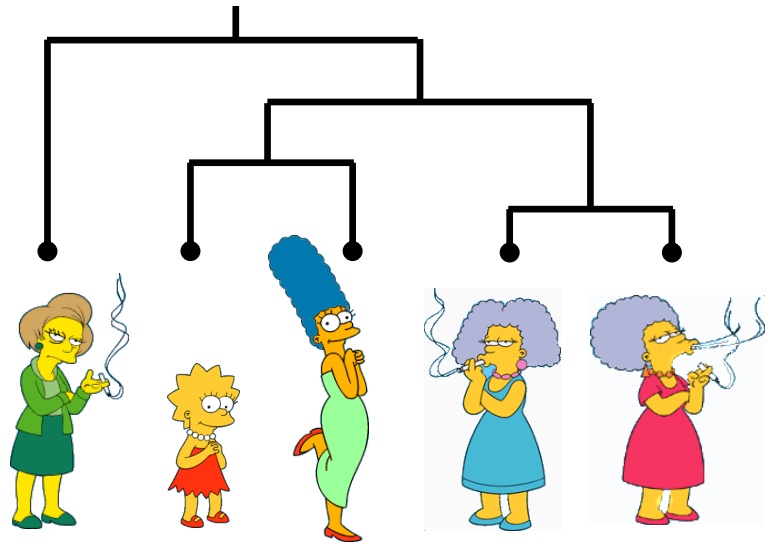
- **Customer segmentation**: Customer segmentation is the practice of dividing a company's customers into groups that reflect similarity among customers in each group
- **Fraud detection**: Using techniques such as K-means Clustering, one can easily identify the patterns of any unusual activities. Detecting an outlier will mean a fraud event has taken place.
- **Document classification**
- **Image segmentation**
- **Anomaly detection**

- A good clustering method will produce high quality clusters with
  - high intra-class similarity
  - low inter-class similarity
- The quality of a clustering result depends on both the similarity measure used by the method and its implementation.
- The quality of a clustering method is also measured by its ability to discover some or all of the hidden patterns.

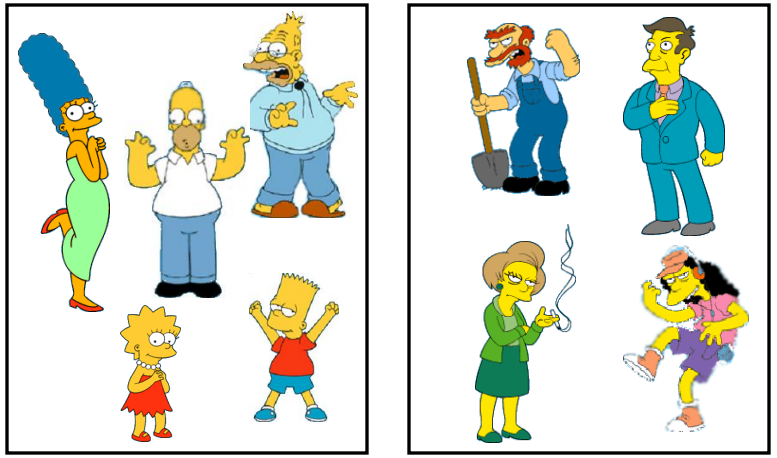
- Hierarchical clustering(BIRCH)
  - A set of nested clusters organized as a hierarchical tree
- Partitional Clustering(k-means,k-mediods)
  - A division data objects into non-overlapping (distinct) subsets (i.e., clusters) such that each data object is in exactly one subset
- Density – Based(DBSCAN)
  - Based on density functions
- Grid-Based(STING)
  - Based on nultiple-level granularity structure
- Model-Based(SOM)
  - Hypothesize a model for each of the clusters and find the best fit of the data to the given model

- Partitional algorithms: Construct various partitions and then evaluate them by some criterion
- Hierarchical algorithms: Create a hierarchical decomposition of the set of objects using some criterion

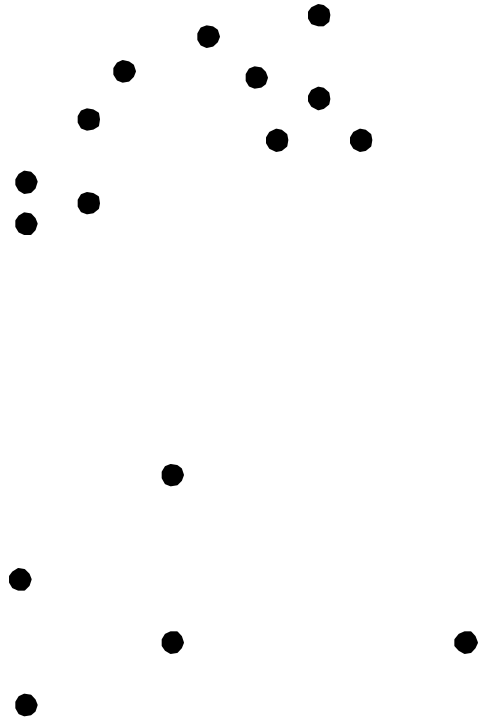
Hierarchical



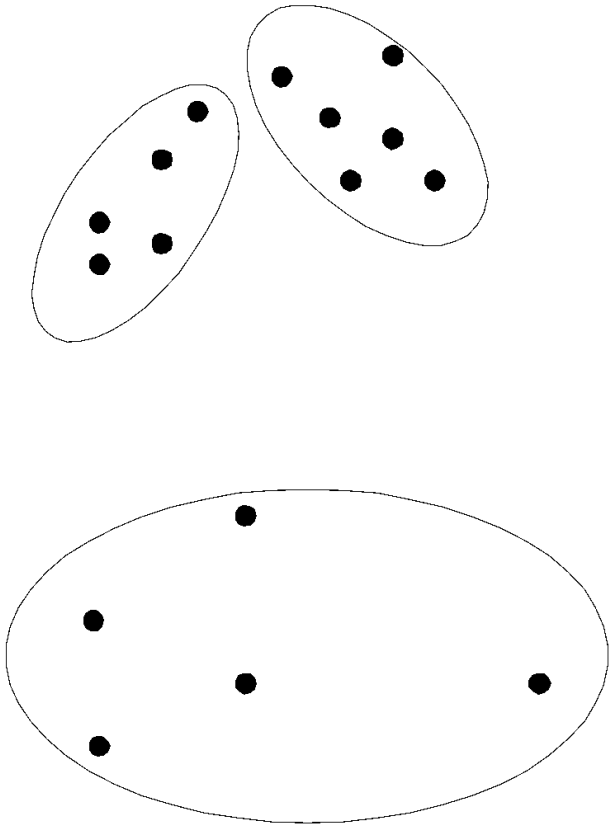
Partitional



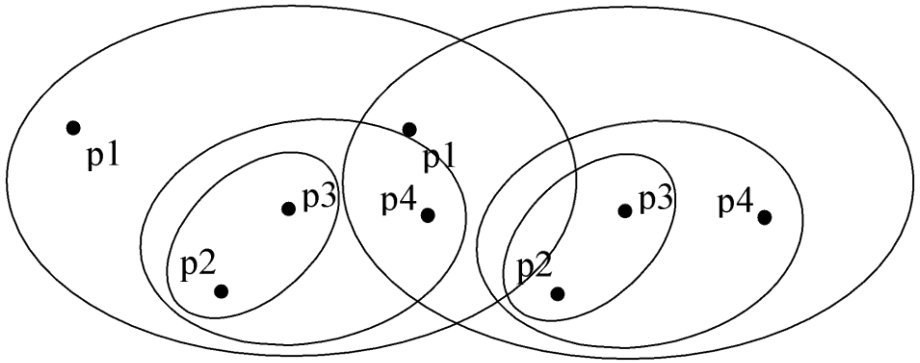
14



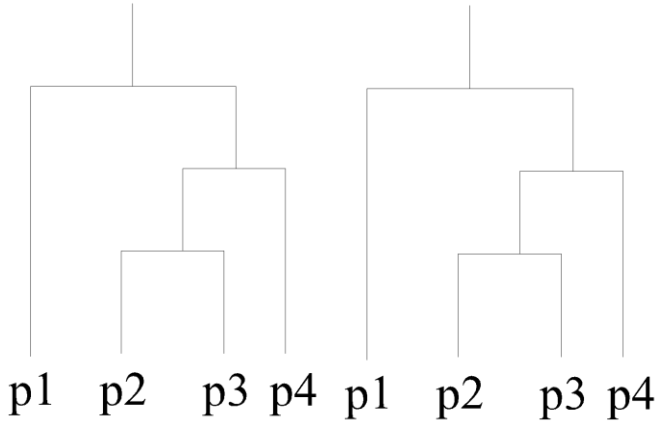
Original Points



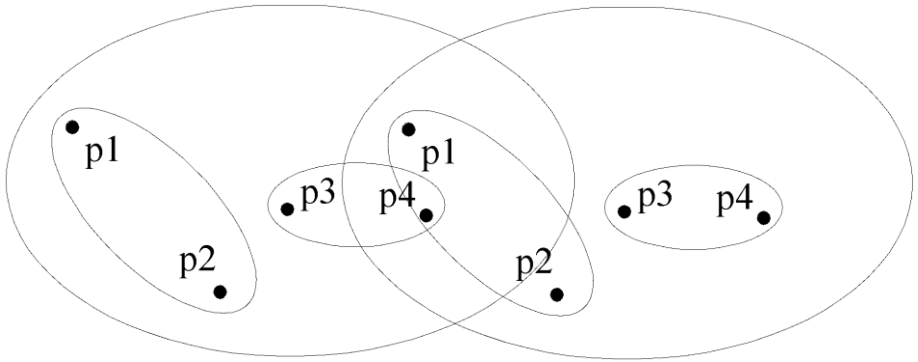
A Partitional Clustering



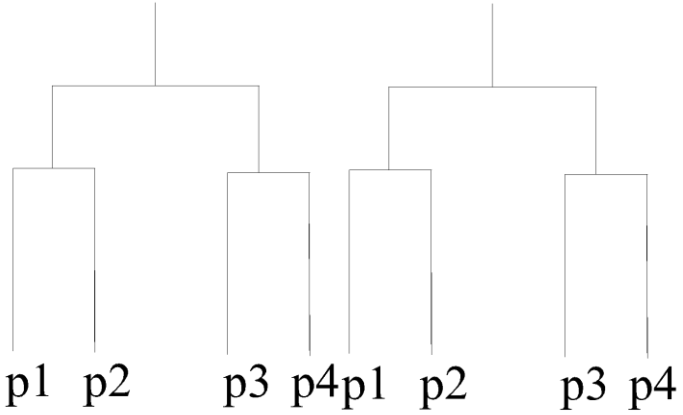
Traditional Hierarchical Clustering



Traditional Dendrogram



Non-traditional Hierarchical Clustering



Non-traditional Dendrogram

- Scalability (in terms of both time and space)
- Ability to deal with different data types
- Minimal requirements for domain knowledge to determine input parameters
- Able to deal with noise and outliers
- Insensitive to order of input records
- Incorporation of user-specified constraints
- Interpretability and usability

Clustering

Clustering is defined as dividing data points or populations into several groups such that similar data points are in the same groups. The aim to segregate groups based on similar traits. Clustering can be divided into two subgroups, broadly:

1. **Soft Clustering**– In this type of clustering, a likelihood or probability of the data point being in a particular cluster is assigned instead of putting each data point into a separate cluster.
2. **Hard Clustering**– Each data point either entirely belongs to a cluster or not at all.

The task of clustering is subjective; i.e., there are many ways of achieving the goal. There can be many different sets of rules for defining similarity among data points. Out of more than 100 clustering algorithms, a few are used correctly. They are as follows: https://www.learndatasci.com/glossary/hierarchical-clustering/

Clustering

1. **Connectivity models**– These models are based on the fact that data points closer in the data space exhibit more similarity than those lying farther away. The model's first approach can classify data points into separate clusters and aggregation as the distance decreases. Another approach is classifying data points as a single cluster and partitioning as the distance increases. The choice of distance function is subjective. The models are easily interpreted but lack scalability for handling large datasets: for example- Hierarchical clustering.
2. **Centroid models**– Iterative clustering algorithms in which similarity is derived as the notion of the closeness of data point to the cluster's centroid. Example- K-Means clustering. The number of clusters is mentioned in advance, which requires prior knowledge of the dataset.

Clustering

3. Distribution models– The models are based on the likelihood of all data points in the cluster belonging to the same distribution. Overfitting is common in these models. Example- Expectation-maximization algorithm

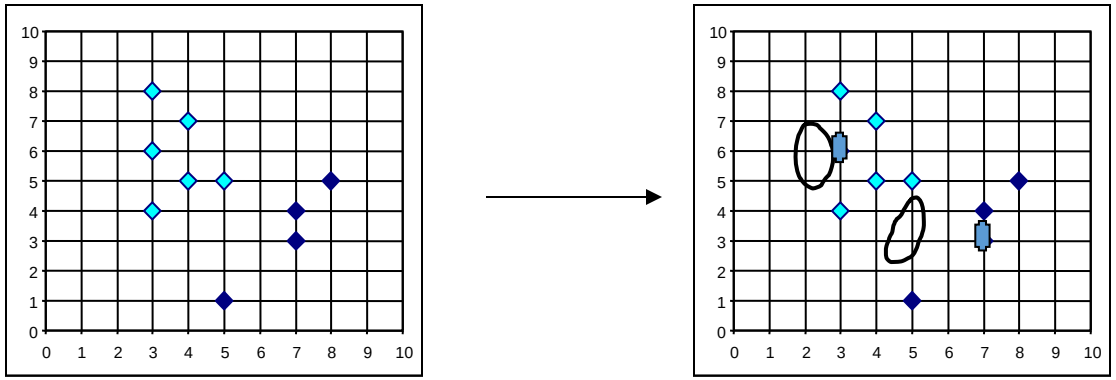
4. Density models– These models search the data space for varying density areas. It isolates various density regions and assigns data points in the same cluster. Example- DBSCAN.

K-Means Clustering

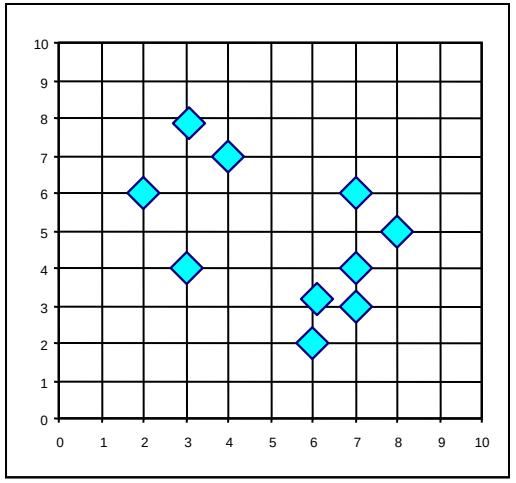
K-Means Clustering

The most common clustering covered in **machine learning for beginners** is K-Means. The first step is to create c new observations among our unlabelled data and locate them randomly, called centroids. The number of centroids represents the number of output classes. The first step of the iterative process for each centroid is to find the nearest point (in terms of Euclidean distance) and assign them to its category. Next, for each category, the average of all the points attributed to that class is computed. The output is the new centroid of the class.

- The k-means algorithm is sensitive to outliers !
  - Since an object with an extremely large value may substantially distort the distribution of the data.
- K-Medoids: Instead of taking the **mean** value of the object in a cluster as a reference point, **medoids** can be used, which is the **most centrally located** object in a cluster.

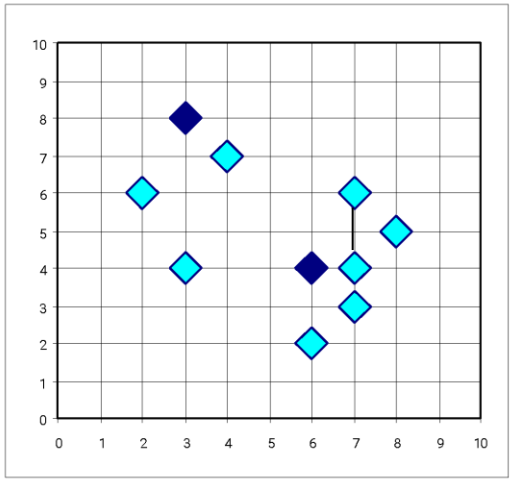


- Find *representative* objects, called medoids, in clusters
- *PAM* (Partitioning Around Medoids, 1987)
  - Starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering
  - *PAM* works effectively for small data sets, but does not scale well for large data sets



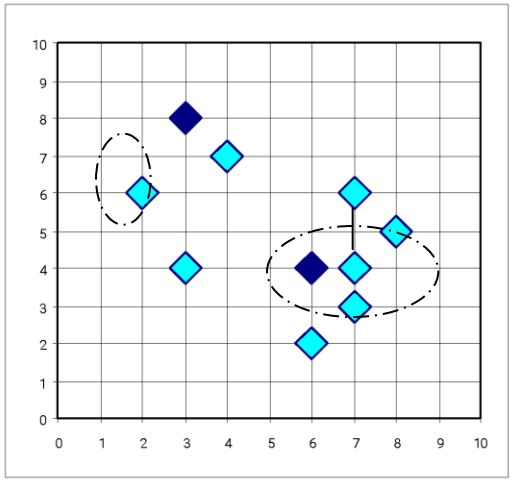
K=2

Arbitrary choose k object as initial medoids

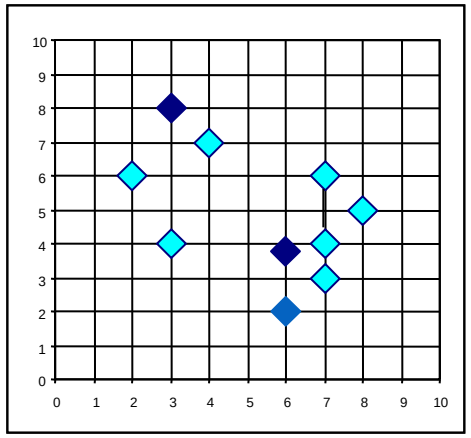


Assign each remaining object to nearest medoids

Total Cost = 20

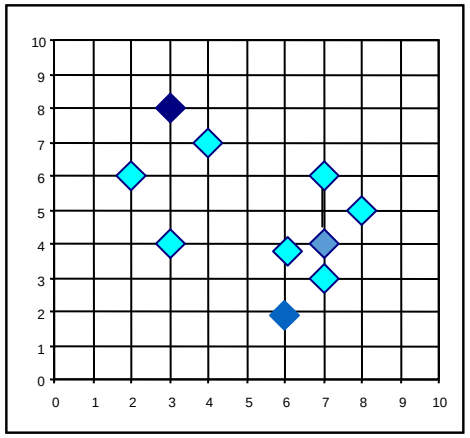


Randomly select a nonmedoid object, O_{ramdom}



Compute total cost of swapping

Total Cost = 26



Swapping O and O_{ramdom}

If quality is improved.

Do loop

Until no change

- Use real object to represent the cluster
  - Select ***k*** representative objects arbitrarily
  - For each pair of non-selected object ***h*** and selected object ***i***, calculate the total swapping cost $\boldsymbol{TC_{ih}}$
  - For each pair of ***i*** and ***h***,
    - If $TC_{ih} < 0$, ***i*** is replaced by ***h***
    - Then assign each non-selected object to the most similar representative object
  - repeat steps 2-3 until there is no change

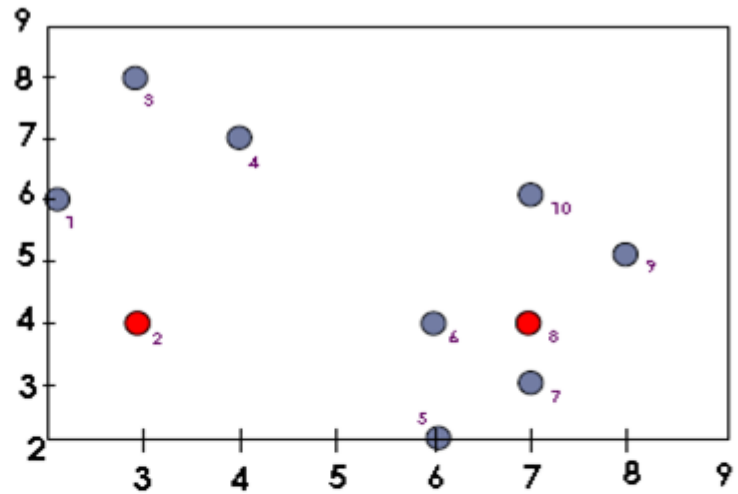
- The iterative process of replacing representative objects by no representative objects continues as long as the quality of the clustering is improved
- For each representative Object O
  - For each non-representative object R, swap O and R
- Choose the configuration with the lowest cost
- Cost function is the difference in absolute error-value if a current representative object is replaced by a non-representative object

- Minimize the sensitivity of k-means to outliers
- Pick actual objects to represent clusters instead of mean values
- Each remaining object is clustered with the representative object (Medoid) to which is the most similar
- The algorithm minimizes the sum of the dissimilarities between each object and its corresponding reference point
  - E: the sum of absolute error for all objects in the data set
  - P: the data point in the space representing an object
  - Oi: is the representative object of cluster Ci

$$E = \sum_{i-1}^{k} \sum_{p \in C_i} | p - o_i |$$

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



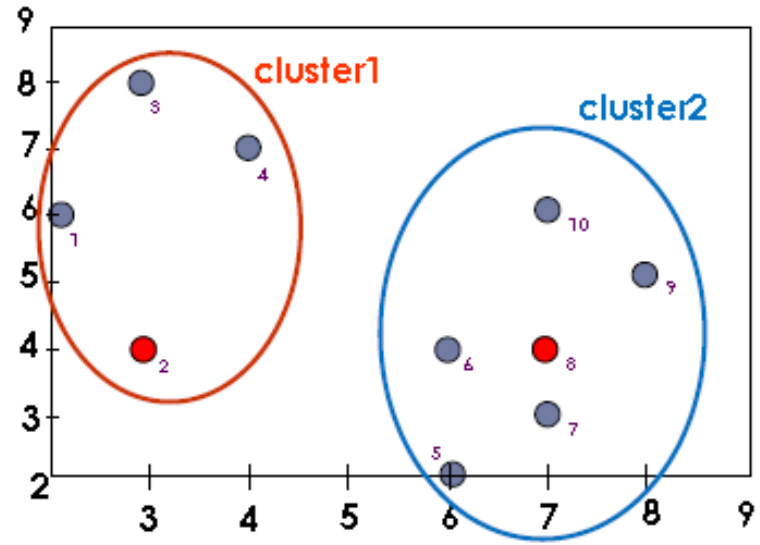
Goal: create two clusters

Choose randmly two medoids

$O_2 = (3,4)$
$O_8 = (7,4)$

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



→ Assign each object to the closest representative object

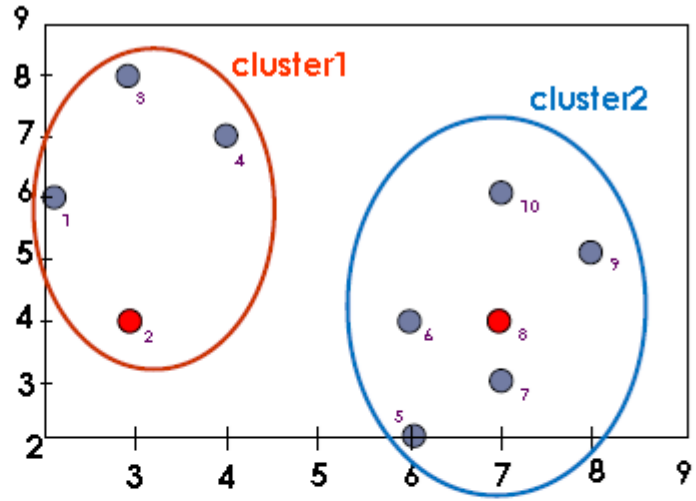
→ Using L1 Metric (Manhattan), we form the following clusters

Cluster1 = $\{O_1, O_2, O_3, O_4\}$

Cluster2 = $\{O_5, O_6, O_7, O_8, O_9, O_{10}\}$

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



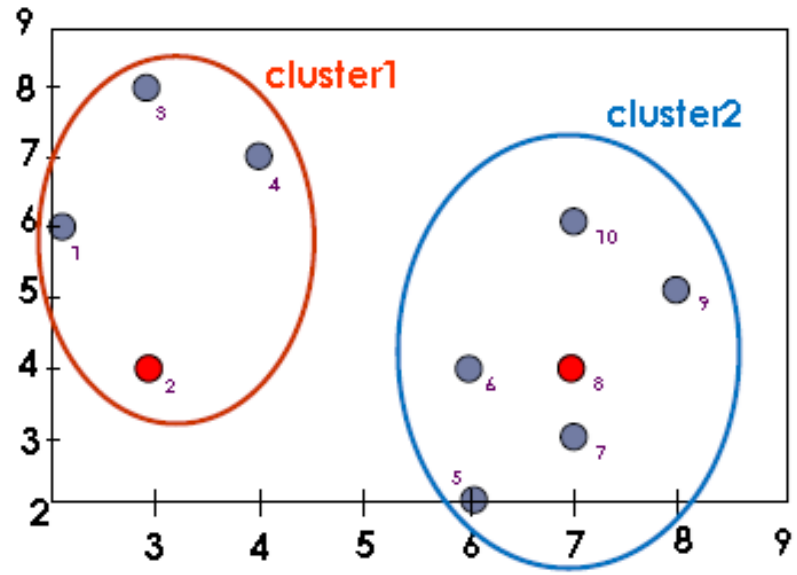
→Compute the absolute error criterion [for the set of Medoids (O2,O8)]

$$E=\sum_{i=1}^{k}\sum_{p\in C_i}|p-o_i|=|o_1-o_2|+|o_3-o_2|+|o_4-o_2|$$

$$+|o_5-o_8|+|o_6-o_8|+|o_7-o_8|+|o_9-o_8|+|o_{10}-o_8|$$

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6

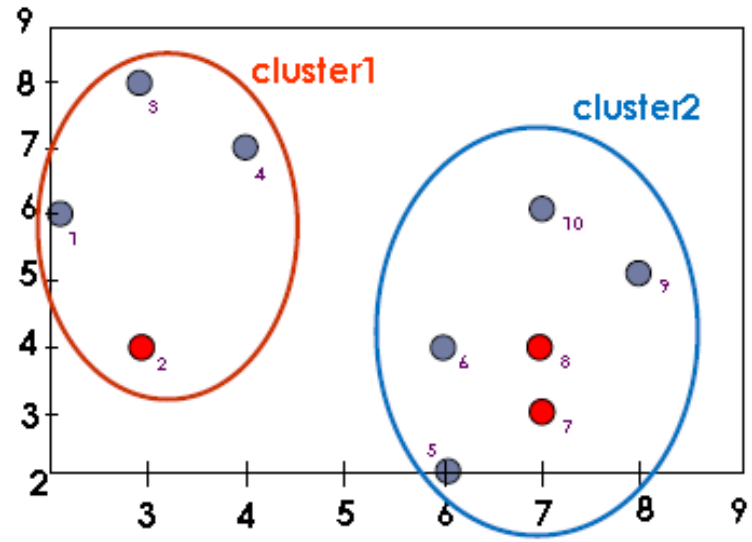


→The absolute error criterion **[for the set of Medoids (O2,O8)]**

$$E = (3+4+4)+(3+1+1+2+2) = 20$$

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



→Choose a random object O_7

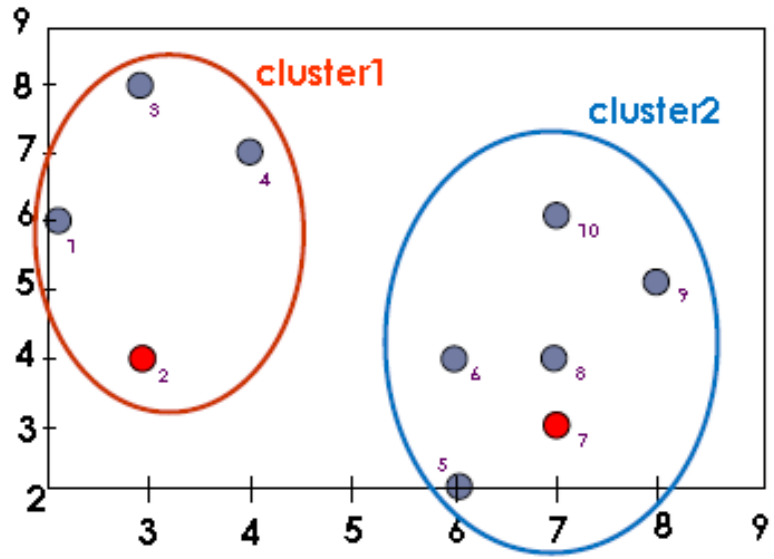
→Swap O8 and O7

→**Compute the absolute error criterion [for the set of Medoids (O2,O7)]**

$$E = (3+4+4)+(2+2+1+3+3) = 22$$

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



→Compute the cost function

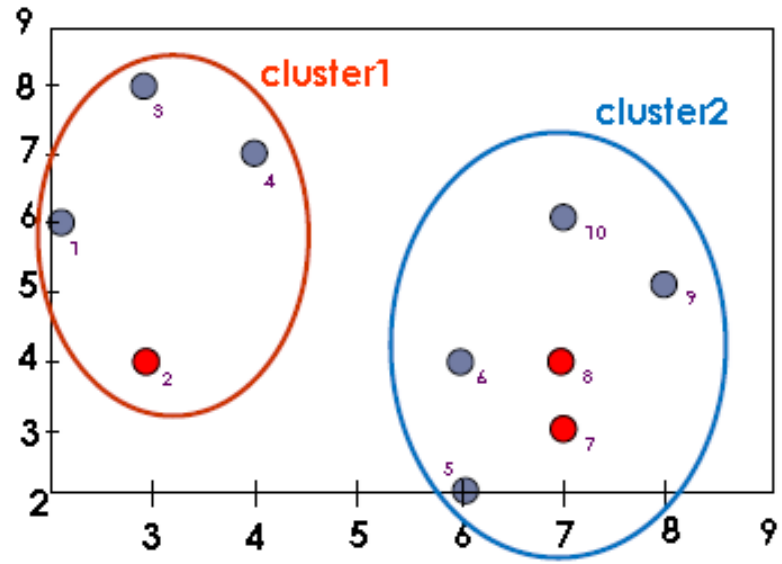
Absolute error [for O_2,O_7] – Absolute error [O_2,O_8]

$$S = 22 - 20$$

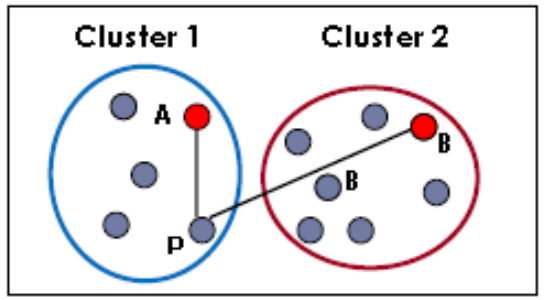
S> 0 ⇒ it is a bad idea to replace O_8 by O_7

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



- In this example, changing the medoid of cluster 2 did not change the assignments of objects to clusters.
- What are the possible cases when we replace a medoid by another object?

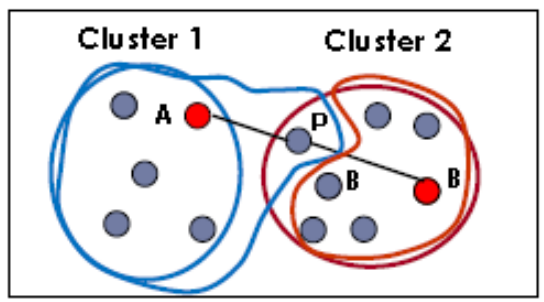


- Representative object
- Random Object

Currently P assigned to A

First case

The assignment of P to A does **not change**

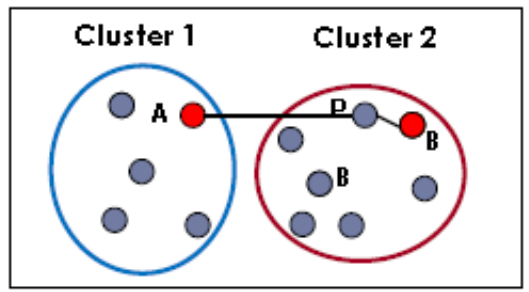


- Representative object
- Random Object

Currently P assigned to B

Second case

P is **reassigned** to A



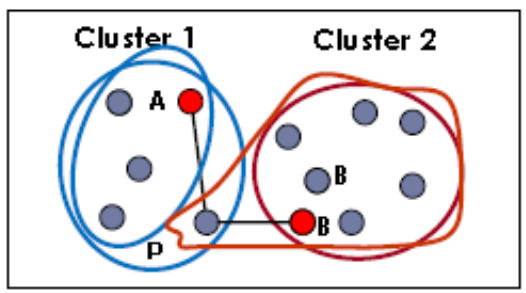
● Representative object

● Random Object

Currently P assigned to B

Third case

P is reassigned to the new B



● Representative object

● Random Object

Currently P assigned to A

Fourth case

P is reassigned to B

- Input
  - K: the number of clusters
  - D: a data set containing n objects
  - Output: A set of k clusters
  - Method:
    1. Arbitrary choose k objects from D as representative objects (seeds)
    2. Repeat (3) Assign each remaining object to the cluster with the nearest representative object
    3. For each representative object Oj
    4. Randomly select a non representative object Orandom
    5. Compute the total cost S of swapping representative object Oj with Orandom (7) if S<0 then replace Oj with Orandom (8) Until no change

- The complexity of each iteration is $O(k(n-k)^2)$
- For large values of n and k, such computation becomes very costly
- Advantages
  - K-Medoids method is more robust than k-Means in the presence of noise and outliers
- Disadvantages
  - K-Medoids is more costly that the k-Means method
  - Like k-means, k-medoids requires the user to specify k
  - It does not scale well for large data sets

K-Medoid Clustering

- **What is K-Medoids Clustering?**
- K-Medoids clustering is an unsupervised machine learning algorithm used to group data into different clusters. It is an iterative algorithm that starts by selecting k data points as medoids in a dataset. After this, the distance between each data point and the medoids is calculated. Then, the data points are assigned to clusters associated with the medoid at the minimum distance from each data point. Here, the medoid is the most centrally located point in the cluster. Once we assign all the data points to the clusters, we calculate the sum of the distance of all the non-medoid data points to the medoid of each cluster. We term the sum of distances as the cost.

K-Medoid Clustering

- **K-Medoids Clustering Algorithm**
- Having an overview of K-Medoids clustering, let us discuss the algorithm for the same.

1. First, we select K random data points from the dataset and use them as medoids.
2. Now, we will calculate the distance of each data point from the medoids. You can use any of the Euclidean, Manhattan distance, or squared Euclidean distances as the distance measure.
3. Once we find the distance of each data point from the medoids, we will assign the data points to the clusters associated with each medoid. The data points are assigned to the medoids at the closest distance.
4. After determining the clusters, we will calculate the sum of the distance of all the non-medoid data points to the medoid of each cluster. Let the cost be C.

K-Medoid Clustering

5. Now, we will select a random data point D_j from the dataset and swap it with a medoid M_i. Here, D_j becomes a temporary medoid. After swapping, we will calculate the distance of all the non-medoid data points to the current medoid of each cluster. Let this cost be C_j.

6. If $C_i > C_j$, the current medoids with D_j as one of the medoids are made permanent medoids. Otherwise, we undo the swap, and M_i is reinstated as the medoid.

7. Repeat 4 to 6 until no change occurs in the clusters.

K-Medoid Clustering

K-Medoids Clustering Example

Point	Coordinates
A1	(2, 6)
A2	(3, 8)
A3	(4, 7)
A4	(6, 2)
A5	(6, 4)
A6	(7, 3)
A7	(7,4)
A8	(8, 5)
A9	(7, 6)
A10	(3, 4)

K-Medoid Clustering

- **Iteration 1**
- Suppose that we want to group the above dataset into two clusters. So, we will randomly choose two medoids.
- Here, the choice of medoids is important for efficient execution. Hence, we have selected two points from the dataset that can be potential medoid for the final clusters. Following are two points from the dataset that we have selected as medoids.
- M1 = (3, 4)
- M2 = (7, 3)
- Now, we will calculate the distance between each data point and the medoids using the Manhattan distance measure. The results have been tabulated as follows.

K-Medoid Clustering

Point	Coordinates	Distance From M1 (3,4)	Distance from M2 (7,3)	Assigned Cluster
A1	(2, 6)	3	8	Cluster 1
A2	(3, 8)	4	9	Cluster 1
A3	(4, 7)	4	7	Cluster 1
A4	(6, 2)	5	2	Cluster 2
A5	(6, 4)	3	2	Cluster 2
A6	(7, 3)	5	0	Cluster 2
A7	(7,4)	4	1	Cluster 2
A8	(8, 5)	6	3	Cluster 2
A9	(7, 6)	6	3	Cluster 2
A10	(3, 4)	0	5	Cluster 1

Iteration-1

K-Medoid Clustering

The clusters made with medoids (3, 4) and (7, 3) are as follows.

Points in cluster1= {(2, 6), (3, 8), (4, 7), (3, 4)}

Points in cluster 2= {(7,4), (6,2), (6, 4), (7,3), (8,5), (7,6)}

After assigning clusters, we will calculate the cost for each cluster and find their sum. The cost is nothing but the sum of distances of all the data points from the medoid of the cluster they belong to.

Hence, the cost for the current cluster will be 3+4+4+2+2+0+1+3+3+0=22.

K-Medoid Clustering

Iteration 2

Now, we will select another non-medoid point (7, 4) and make it a temporary medoid for the second cluster. Hence,

M1 = (3, 4)

M2 = (7, 4)

Now, let us calculate the distance between all the data points and the current medoids.

K-Medoid Clustering

Point	Coordinates	Distance From M1 (3,4)	Distance from M2 (7,4)	Assigned Cluster
A1	(2, 6)	3	7	Cluster 1
A2	(3, 8)	4	8	Cluster 1
A3	(4, 7)	4	6	Cluster 1
A4	(6, 2)	5	3	Cluster 2
A5	(6, 4)	3	1	Cluster 2
A6	(7, 3)	5	1	Cluster 2
A7	(7,4)	4	0	Cluster 2
A8	(8, 5)	6	2	Cluster 2
A9	(7, 6)	6	2	Cluster 2
A10	(3, 4)	0	4	Cluster 1

K-Medoid Clustering

The data points haven't changed in the clusters after changing the medoids.
Hence, clusters are:
Points in cluster1:{(2, 6), (3, 8), (4, 7), (3, 4)}
Points in cluster 2:{(7,4), (6,2), (6, 4), (7,3), (8,5), (7,6)}
Now, let us again calculate the cost for each cluster and find their sum. The total cost this time will be 3+4+4+3+1+1+0+2+2+0=20.
Here, the current cost is less than the cost calculated in the previous iteration. Hence, we will make the swap permanent and make (7,4) the medoid for cluster 2. If the cost this time was greater than the previous cost i.e. 22, we would have to revert the change. New medoids after this iteration are (3, 4) and (7, 4) with no change in the clusters.

K-Medoid Clustering

Iteration 3

Now, let us again change the medoid of cluster 2 to (6, 4). Hence, the new medoids for the clusters are M1=(3, 4) and M2= (6, 4). Let us calculate the distance between the data points and the above medoids to find the new cluster. The results have been tabulated as follows.

K-Medoid Clustering

Point	Coordinates	Distance From M1 (3,4)	Distance from M2 (6,4)	Assigned Cluster
A1	(2, 6)	3	6	Cluster 1
A2	(3, 8)	4	7	Cluster 1
A3	(4, 7)	4	5	Cluster 1
A4	(6, 2)	5	2	Cluster 2
A5	(6, 4)	3	0	Cluster 2
A6	(7, 3)	5	2	Cluster 2
A7	(7,4)	4	1	Cluster 2
A8	(8, 5)	6	3	Cluster 2
A9	(7, 6)	6	3	Cluster 2
A10	(3, 4)	0	3	Cluster 1

Again, the clusters haven't changed. Hence, clusters are:

Points in cluster1:{(2, 6), (3, 8), (4, 7), (3, 4)}

Points in cluster 2:{(7,4), (6,2), (6, 4), (7,3), (8,5), (7,6)}

Now, let us again calculate the cost for each cluster and find their sum. The total cost this time will be 3+4+4+2+0+2+1+3+3+0=22.

The current cost is 22 which is greater than the cost in the previous iteration i.e. 20. Hence, we will revert the change and the point (7, 4) will again be made the medoid for cluster 2.

So, the clusters after this iteration will be cluster1 = {(2, 6), (3, 8), (4, 7), (3, 4)} and cluster 2= {(7,4), (6,2), (6, 4), (7,3), (8,5), (7,6)}. The medoids are (3,4) and (7,4).

Replacing the medoids with a non-medoid data point. The set of medoids for which the cost is the least, the medoids, and the associated clusters are made permanent. So, after all the iterations, you will get the final clusters and their medoids.

The K-Medoids clustering algorithm is a computation-intensive algorithm that requires many iterations. In each iteration, we need to calculate the distance between the medoids and the data points, assign clusters, and compute the cost. Hence, K-Medoids clustering is not well suited for large data sets.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Advantages of K-Medoids Clustering

- K-Medoids clustering is a simple iterative algorithm and is very easy to implement.
- K-Medoids clustering is guaranteed to converge. Hence, we are guaranteed to get results when we perform k-medoids clustering on any dataset.
- K-Medoids clustering doesn't apply to a specific domain. Owing to the generalization, we can use k-medoids clustering in different machine learning applications ranging from text data to Geo-spatial data and financial data to e-commerce data.
- The medoids in k-medoids clustering are selected randomly. We can choose the initial medoids in a way such that the number of iterations can be minimized. To improve the performance of the algorithm, you can warm start the choice of medoids by selecting specific data points as medoids after data analysis.
- Compared to other partitioning clustering algorithms such as K-median and k-modes clustering, the k-medoids clustering algorithm is faster in execution.
- K-Medoids clustering algorithm is very robust and it effectively deals with outliers and noise in the dataset. Compared to k-means clustering, the k-medoids clustering algorithm is a better choice for analyzing data with significant noise and outliers.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Disadvantages of K-Medoids Clustering

- K-Medoids clustering is not a scalable algorithm. For large datasets, the execution time of the algorithm becomes very high. Due to this, you cannot use the K-Medoids algorithm for very large datasets.
- In K-Medoids clustering, we don't know the optimal number of clusters. Hence, we need to perform clustering using different values of k to find the optimal number of clusters.
- When the dimensionality in the dataset increases, the distance between the data points starts to become similar. Due to this, the distance between a data point and various medoids becomes almost the same. This introduces inefficiency in the execution. To overcome this problem, you can use advanced clustering algorithms like spectral clustering. Alternatively, you can also try to reduce the dimensionality of the dataset while data preprocessing.
- K-Medoids clustering algorithm randomly chooses the initial medoids. Also, the output clusters are primarily dependent on the initial medoids. Due to this, every time you run the k-medoids clustering algorithm, you will get unique clusters.
- The K-Medoids clustering algorithm uses the distance from a central point (medoid). Due to this, the k-medoids algorithm gives circular/spherical clusters. This algorithm is not useful for clustering data into arbitrarily shaped clusters.

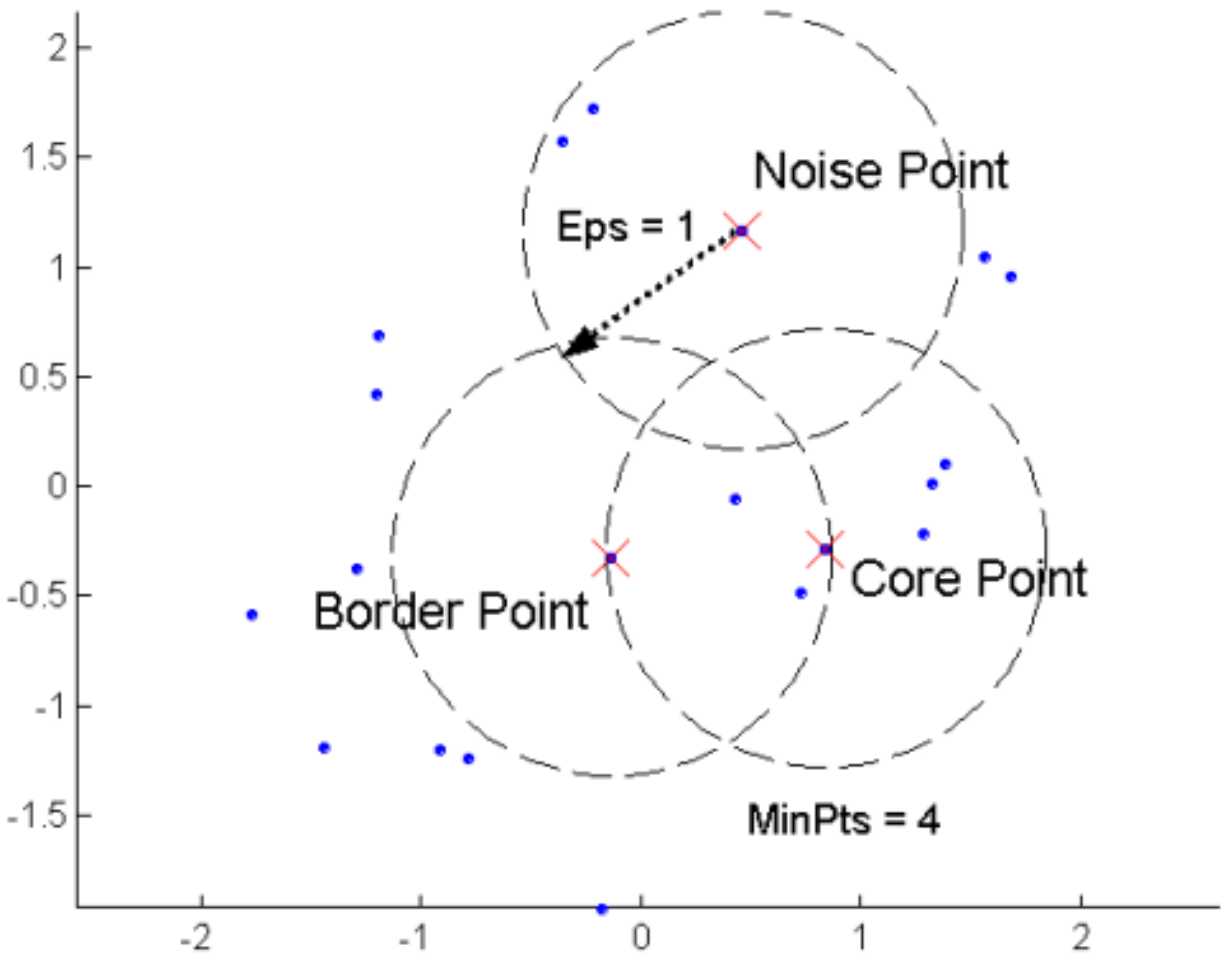
DBSCAN: Density-Based Clustering

- DBSCAN is a Density-Based Clustering algorithm
- Reminder: In density based clustering we partition points into dense regions separated by not-so-dense regions.
- Important Questions:
  - How do we measure density?
  - What is a dense region?
- DBSCAN:
  - Density at point p: number of points within a circle of radius Eps
  - Dense Region: A circle of radius Eps that contains at least MinPts points

DBSCAN

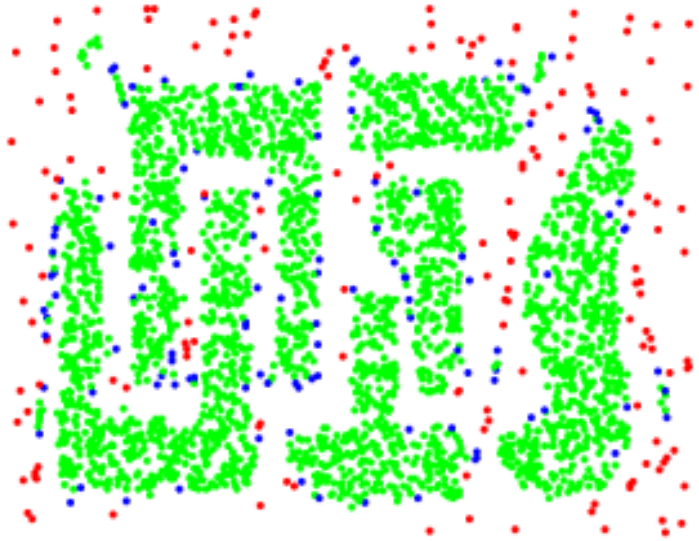
- Characterization of points
  - A point is a core point if it has more than a specified number of points (MinPts) within Eps
    - These points belong in a dense region and are at the interior of a cluster
  - A border point has fewer than MinPts within Eps, but is in the neighborhood of a core point.
  - A noise point is any point that is not a core point or a border point.

DBSCAN: Core, Border, and Noise Points



DBSCAN: Core, Border and Noise Points

Original Points

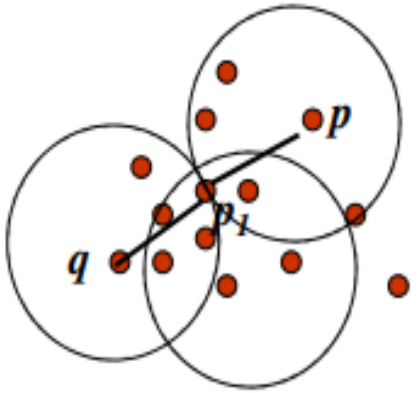


Point types: core, border and noise

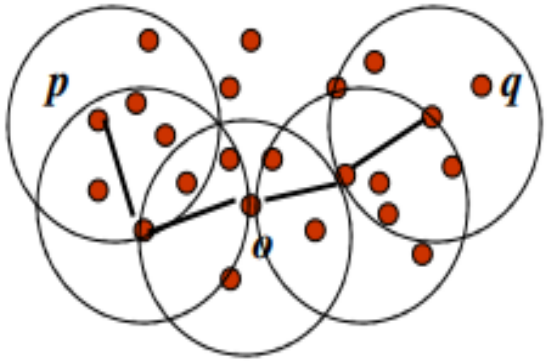
Eps = 10, MinPts = 4

Density-Connected points

- Density edge
  - We place an edge between two core points q and p if they are within distance Eps.



- Density-connected
  - A point p is density-connected to a point q if there is a path of edges from p to q

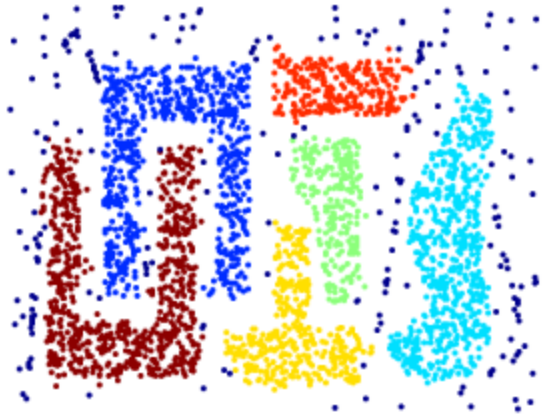


DBSCAN Algorithm

- Label points as core, border and noise
- Eliminate noise points
- For every core point p that has not been assigned to a cluster
  - Create a new cluster with the point p and all the points that are density-connected to p.
- Assign border points to the cluster of the closest core point.

When DBSCAN Works Well

Original Points

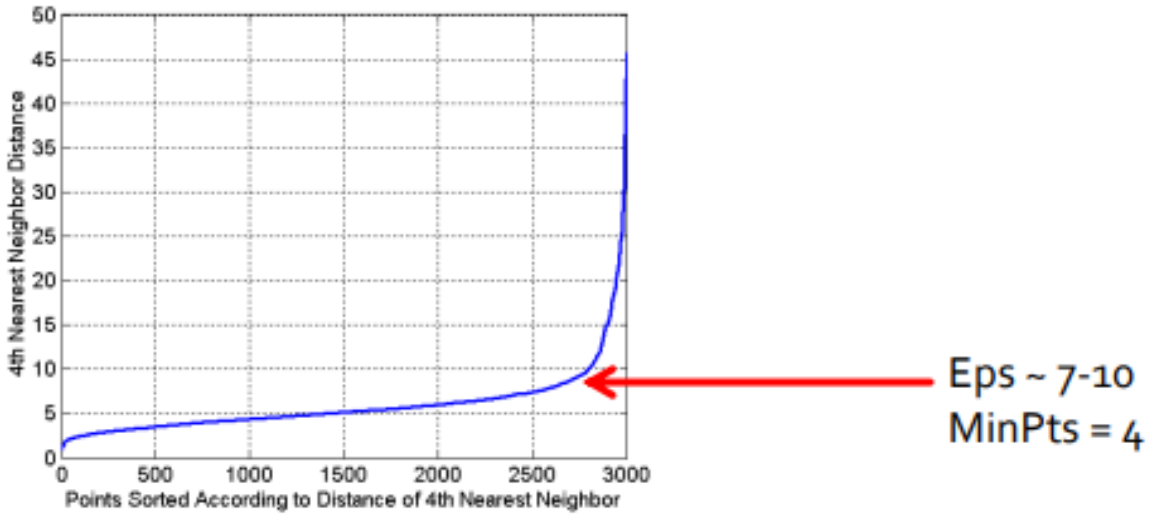


Clusters

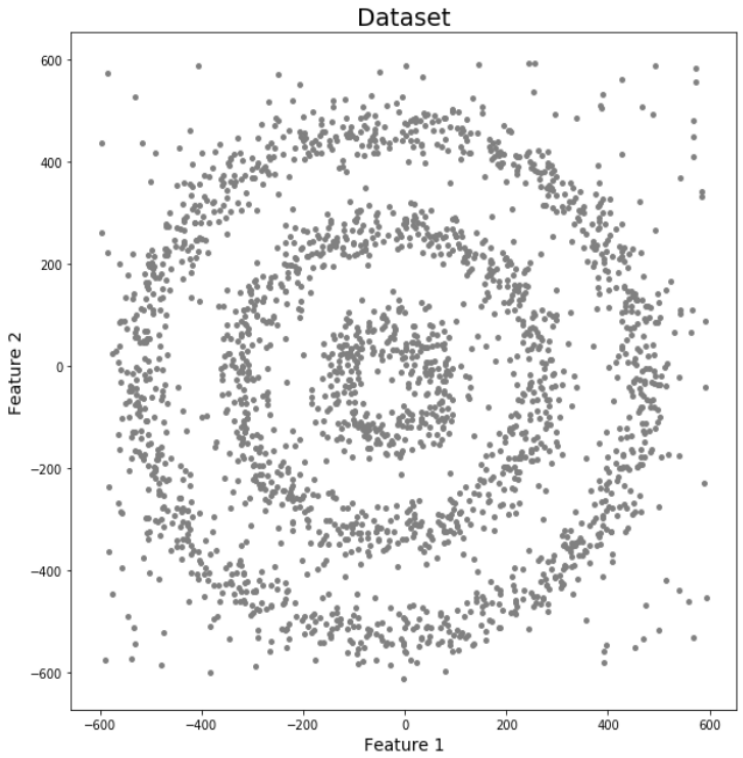
- Resistant to Noise
- Can handle clusters of different shapes and sizes

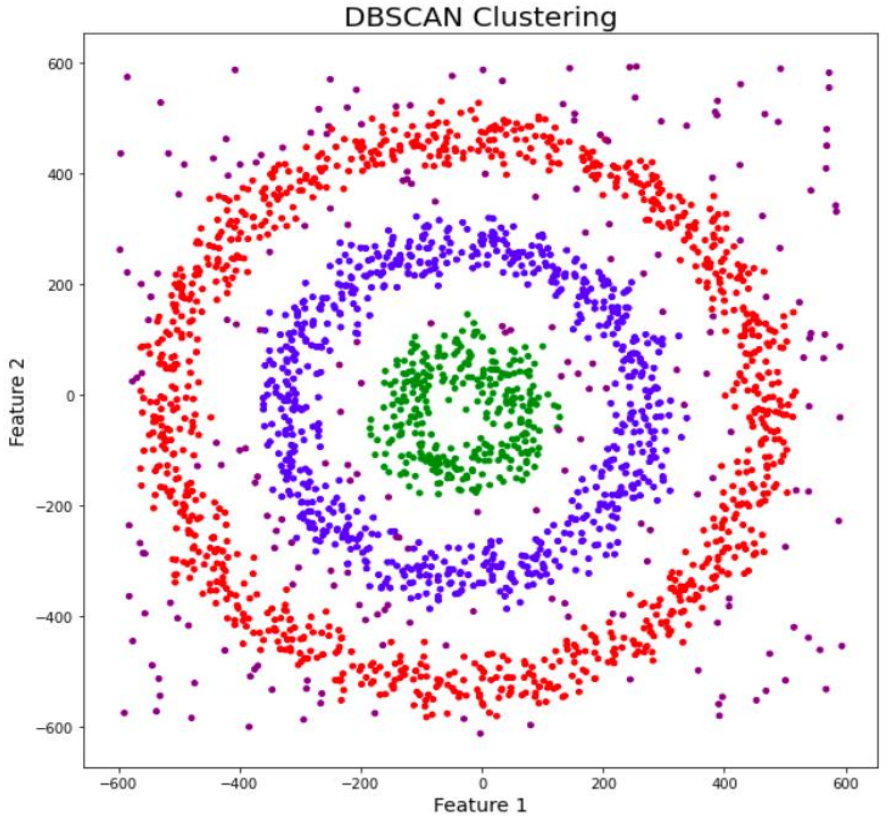
DBSCAN: Determining Eps and MinPts

- Idea is that for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance
- Noise points have the k^{th} nearest neighbor at farther distance
- So, plot sorted distance of every point to its k^{th} nearest neighbor
- Find the distance d where there is a "knee" in the curve
  - Eps = d, MinPts = k



Clustering -Example





What Exactly is DBSCAN Clustering?

DBSCAN stands for **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise

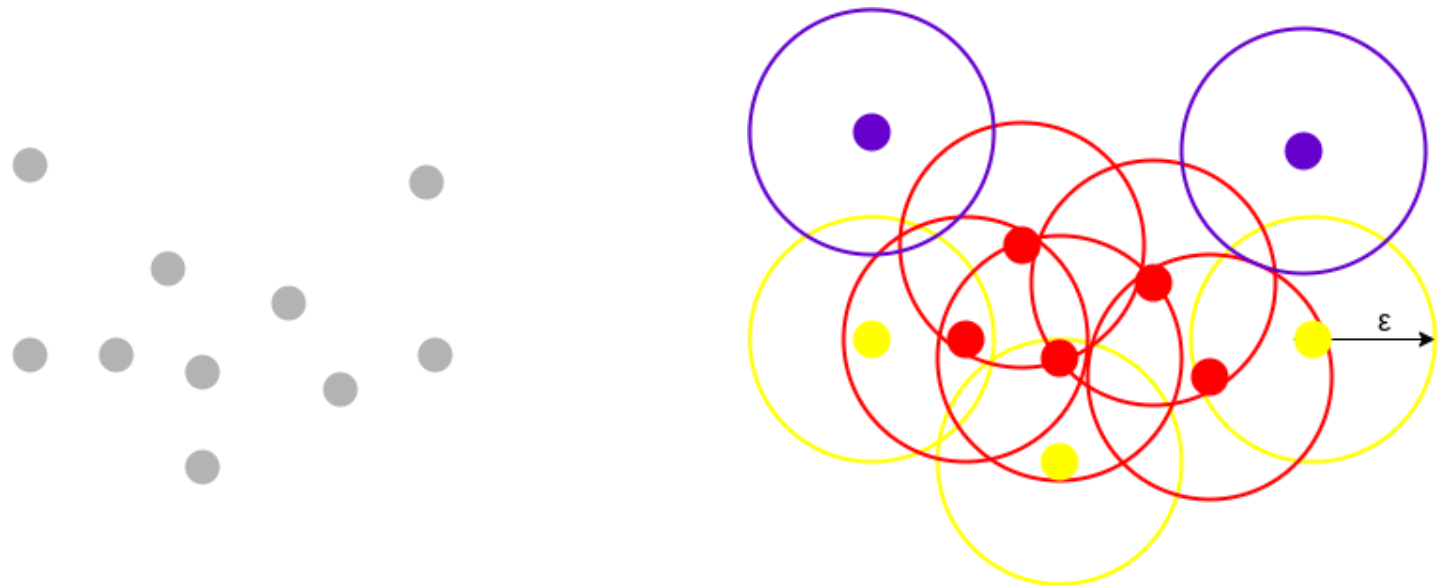
It groups 'densely grouped' data points into a single cluster. It can identify clusters in large spatial datasets by looking at the local density of the data points. **The most exciting feature of DBSCAN clustering is that it is robust to outliers**. It also does not require the number of clusters to be told beforehand, unlike K-Means, where we have to specify the number of centroids.

DBSCAN requires only two parameters: *epsilon* and *minPoints*. ***Epsilon*** is the radius of the circle to be created around each data point to check the density and ***minPoints*** is the minimum number of data points required inside that circle for that data point to be classified as a **Core** point.

In higher dimensions the circle becomes hypersphere, *epsilon* becomes the radius of that hypersphere, and *minPoints* is the minimum number of data points required inside that hypersphere.

DBSCAN Clustering

DBSCAN creates a circle of *epsilon* radius around every data point and classifies them into **Core** point, **Border** point, and **Noise**. A data point is a **Core** point if the circle around it contains at least '*minPoints*' number of points. If the number of points is less than *minPoints*, then it is classified as **Border** Point, and if there are no other data points around any data point within *epsilon* radius, then it treated as **Noise**.



The above figure shows us a cluster created by DBCAN with *minPoints = 3*. Here, we draw a circle of equal radius *epsilon* around every data point. These two parameters help in creating spatial clusters.

DBSCAN Clustering

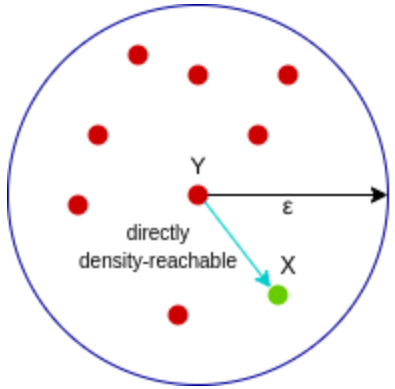
All the data points with at least 3 points in the circle including itself are considered as **Core** points represented by red color. All the data points with less than 3 but greater than 1 point in the circle including itself are considered as **Border** points. They are represented by yellow color. Finally, data points with no point other than itself present inside the circle are considered as **Noise** represented by the purple color.

For locating data points in space, DBSCAN uses Euclidean distance, although other methods can also be used (like great circle distance for geographical data). It also needs to scan through the entire dataset once, whereas in other algorithms we have to do it multiple times.

Reachability and Connectivity

These are the two concepts that you need to understand before moving further. Reachability states if a data point can be accessed from another data point directly or indirectly, whereas Connectivity states whether two data points belong to the same cluster or not. In terms of reachability and connectivity, two points in DBSCAN can be referred to as:

- **Directly Density-Reachable**
- **Density-Reachable**
- **Density-Connected**



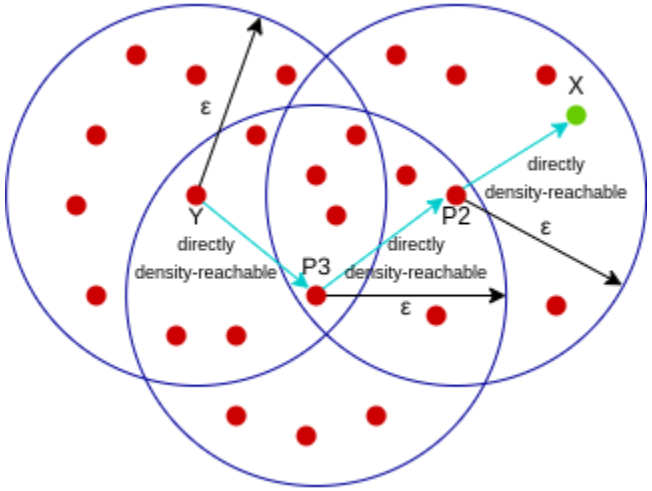
A point **X** is **directly density-reachable** from point **Y** w.r.t *epsilon, minPoints* if,

1. **X** belongs to the neighborhood of **Y**, i.e, *dist(X, Y) <= epsilon*
2. **Y** is a core point

Here, X is directly density-reachable from Y, but vice versa is not valid.

DBSCAN Clustering

A point **X** is **density-reachable** from point **Y** w.r.t *epsilon, minPoints* if there is a chain of points p1, p2, p3, ..., pn and p1=**X** and pn=**Y** such that pi+1 is directly density-reachable from pi.

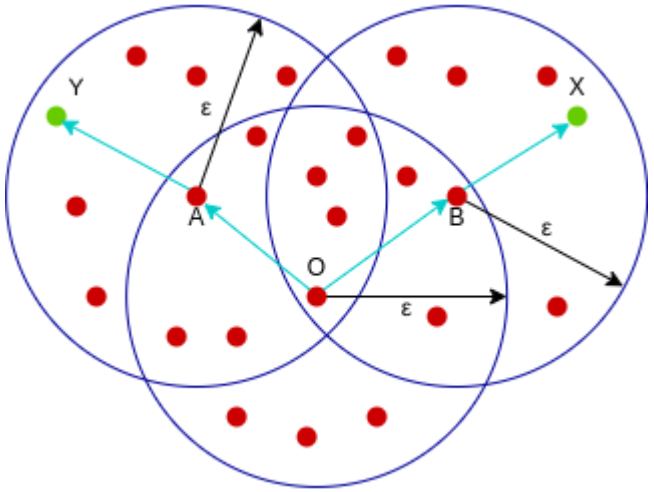


Here, **X** is density-reachable from **Y** with **X** being directly density-reachable from **P2**, **P2** from **P3,** and **P3** from **Y.**

But, the inverse of this is not valid.

DBSCAN Clustering

A point **X** is **density-connected** from point **Y** w.r.t *epsilon and minPoints* if there exists a point **O** such that both **X** and **Y** are density-reachable from **O** w.r.t to *epsilon and minPoints.*



Here, both **X** and **Y** are density-reachable from **O**, therefore, we can say that **X** is density-connected from **Y**.

Parameter Selection in DBSCAN Clustering

DBSCAN is very sensitive to the values of *epsilon* and *minPoints*. Therefore, it is very important to understand how to select the values of *epsilon* and *minPoints*. A slight variation in these values can significantly change the results produced by the DBSCAN algorithm.

The value of *minPoints* should be at least one greater than the number of dimensions of the dataset, i.e.,

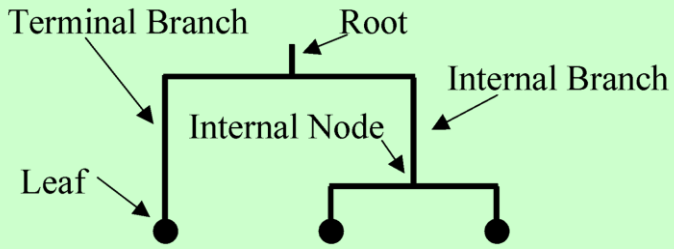
$$minPoints >= Dimensions + 1.$$

It does not make sense to take *minPoints* as 1 because it will result in each point being a separate cluster. Therefore, it must be at least 3. Generally, it is twice the dimensions. But domain knowledge also decides its value.

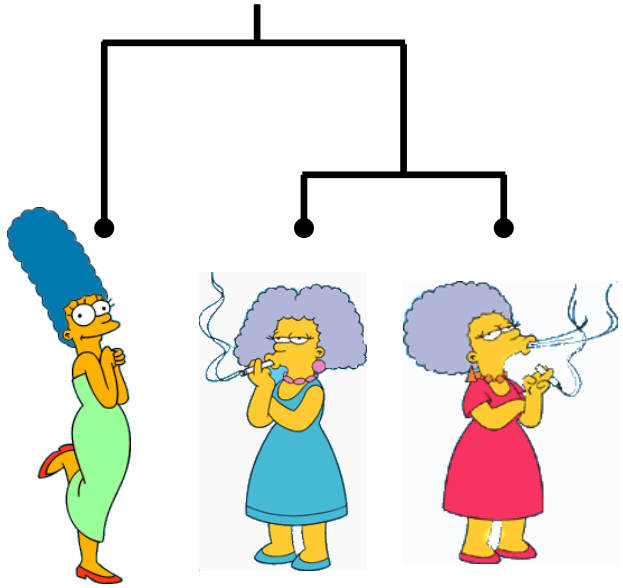
If the value of *epsilon* chosen is too small then a higher number of clusters will be created, and more data points will be taken as noise. Whereas, if chosen too big then various small clusters will merge into a big cluster, and we will lose details.

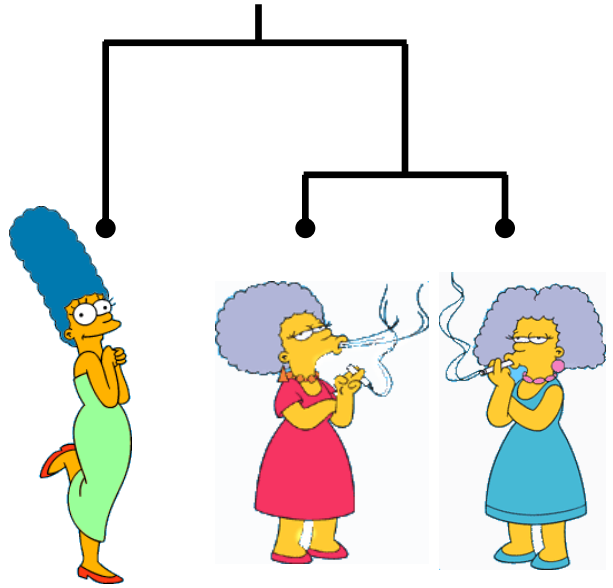
Hierarchical Clustering

Dendrogram -A Useful Tool for Summarizing Similarity Measurements



The similarity between two objects in a dendrogram is represented as the height of the lowest internal node they share.

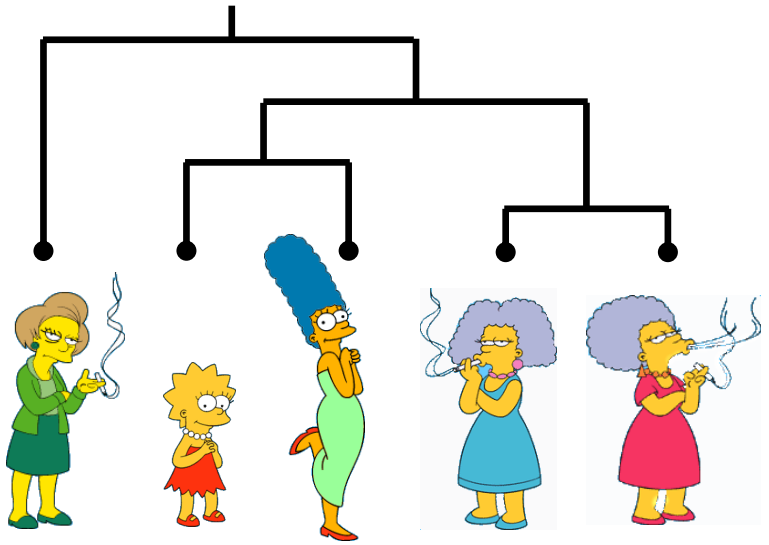




Hierarchical Clustering

The number of dendrograms with n leafs $= (2n-3)!/[(2^{(n-2)})(n-2)!]$

Number of Leafs	Number of Possible Dendrograms
2	1
3	3
4	15
5	105
...	…
10	34,459,425

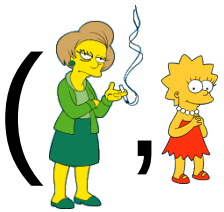


Since we cannot test all possible trees we will have to heuristic search of all possible trees. We could do this..

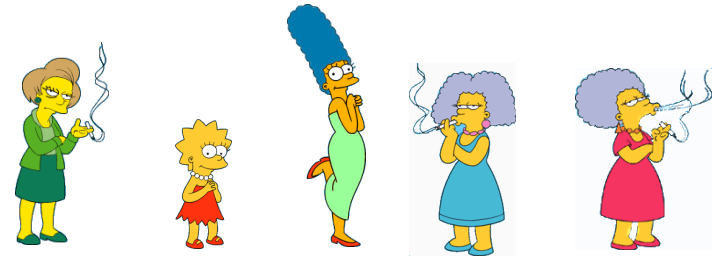
Bottom-Up (agglomerative): Starting with each item in its own cluster, find the best pair to merge into a new cluster. Repeat until all clusters are fused together.

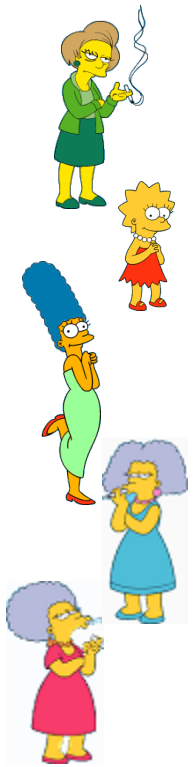
Top-Down (divisive): Starting with all the data in a single cluster, consider every possible way to divide the cluster into two. Choose the best division and recursively operate on both sides.

We begin with a distance matrix which contains the distances between every pair of objects in our database.

D() = 8

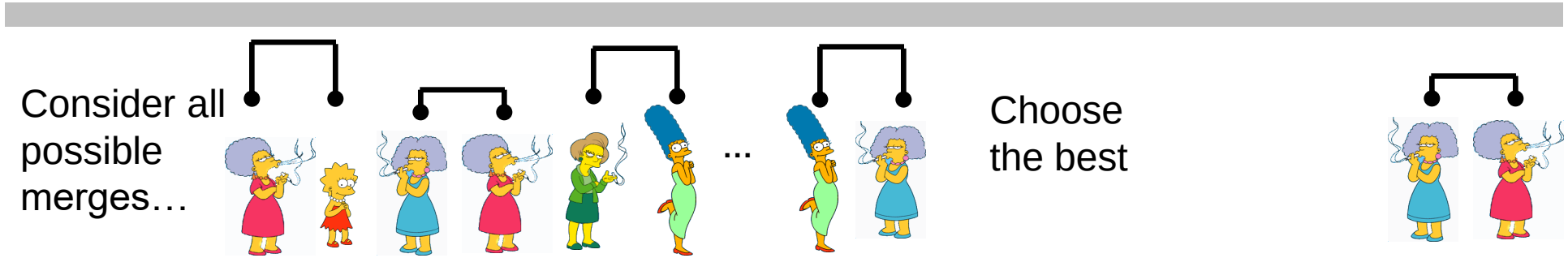
D() = 1



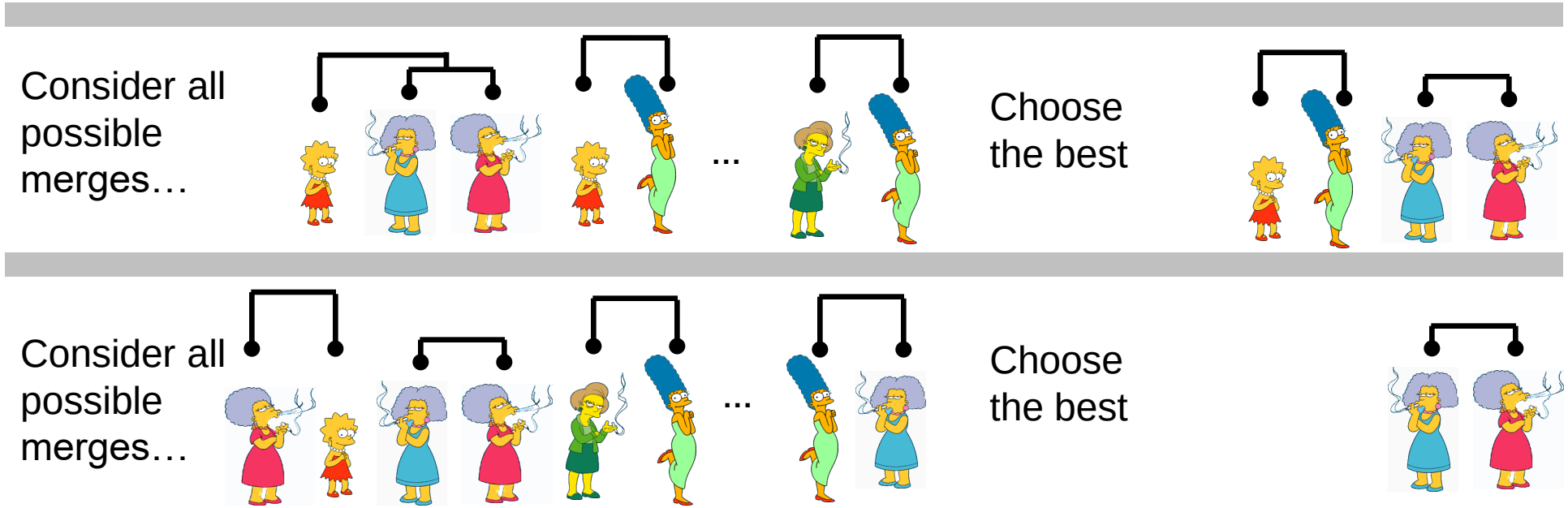


0	8	8	7	7
	0	2	4	4
		0	3	3
			0	1
				0

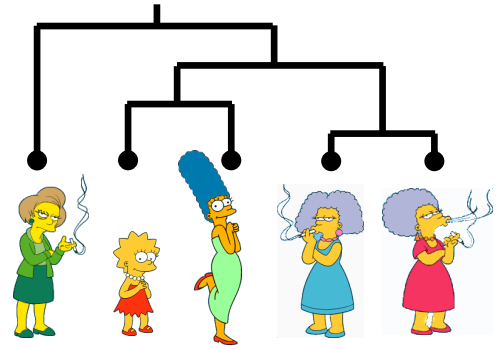
Bottom-Up (agglomerative): Starting with each item in its own cluster, find the best pair to merge into a new cluster. Repeat until all clusters are fused together.



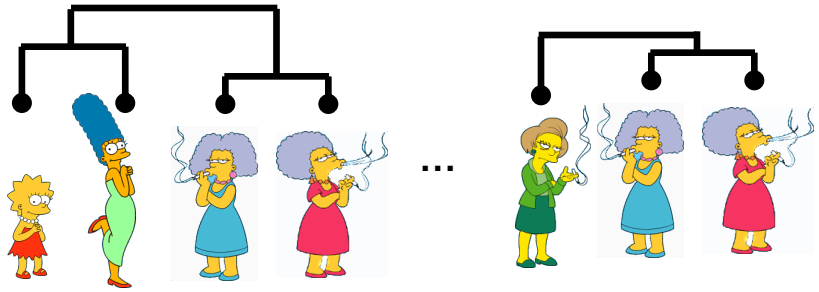
Bottom-Up (agglomerative): Starting with each item in its own cluster, find the best pair to merge into a new cluster. Repeat until all clusters are fused together.



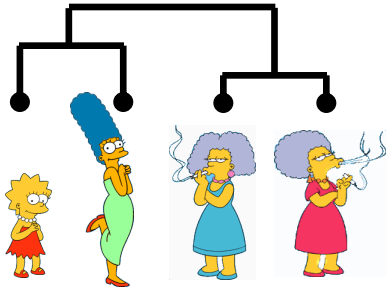
Bottom-Up (agglomerative): Starting with each item in its own cluster, find the best pair to merge into a new cluster. Repeat until all clusters are fused together.



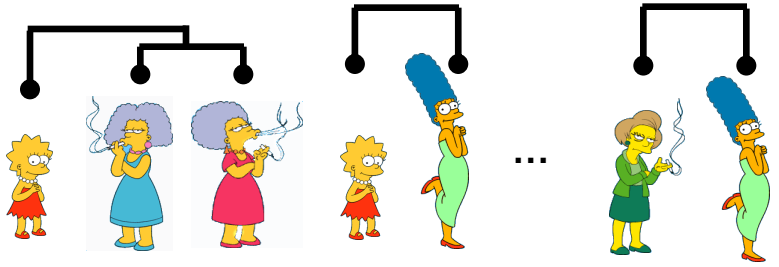
Consider all possible merges…



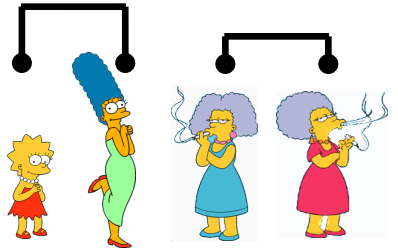
Choose the best



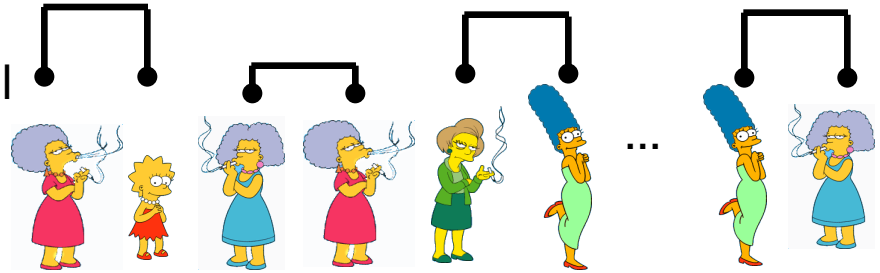
Consider all possible merges…



Choose the best

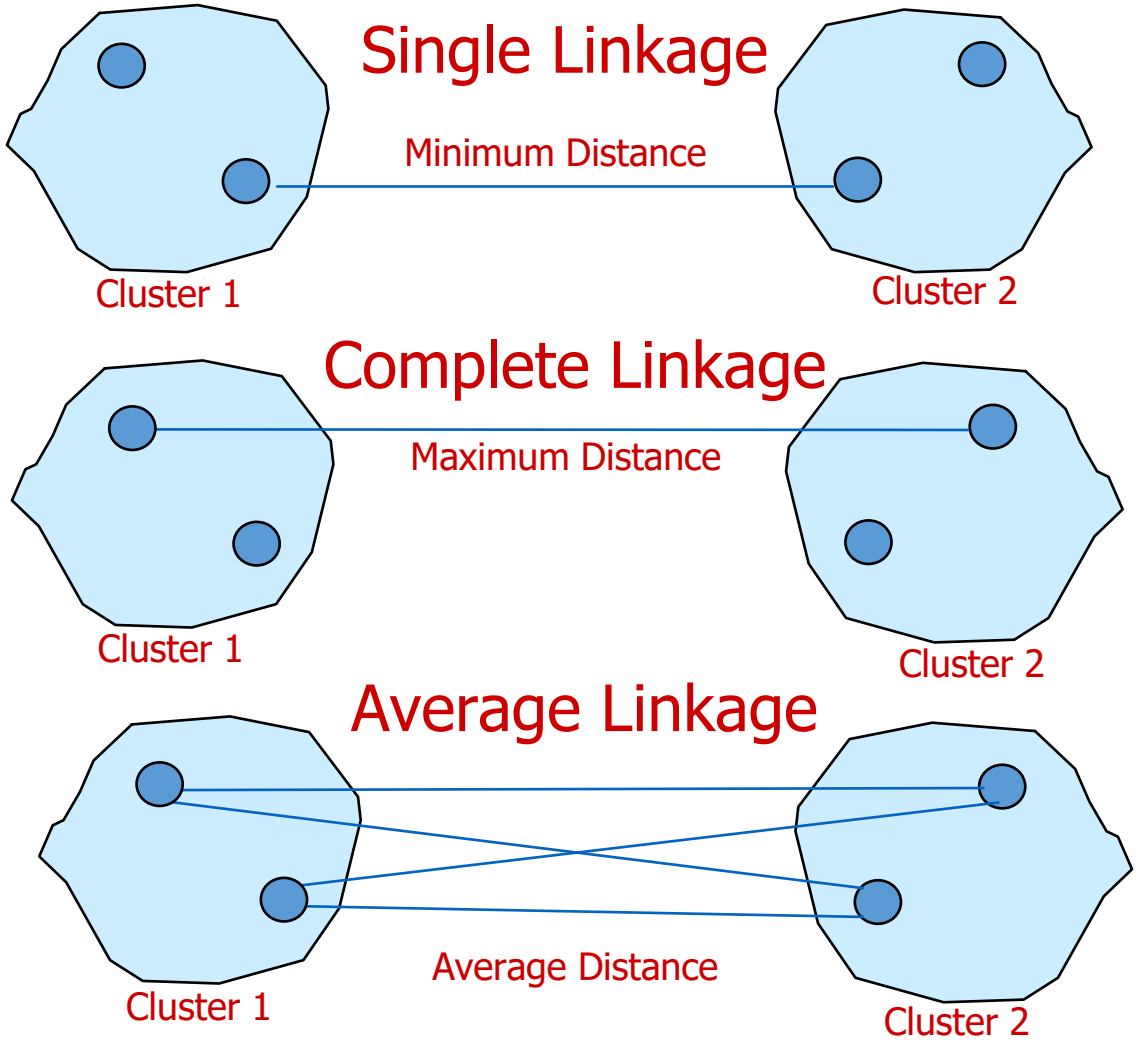


Consider all possible merges…



Choose the best

- The **single linkage** method is based on minimum distance, or the nearest neighbor rule.
- The **complete linkage** method is based on the maximum distance or the furthest neighbor approach.
- The **average linkage** method the distance between two clusters is defined as the average of the distances between all pairs of objects


Single Linkage
Minimum Distance
Cluster 1
Cluster 2
Complete Linkage
Maximum Distance
Cluster 1
Cluster 2
Average Linkage
Average Distance
Cluster 1
Cluster 2

Dist	A	B	C	D	E	F
A	0.00	0.71	5.66	3.61	4.24	3.20
B	0.71	0.00	4.95	2.92	3.54	2.50
C	5.66	4.95	0.00	2.24	1.41	2.50
D	3.61	2.92	2.24	0.00	1.00	0.50
E	4.24	3.54	1.41	1.00	0.00	1.12
F	3.20	2.50	2.50	0.50	1.12	0.00

Min Distance (Single Linkage)

Dist	A	B	C	D, F	E
A	0.00	0.71	5.66	?	4.24
B	0.71	0.00	4.95	?	3.54
C	5.66	4.95	0.00	?	1.41
D, F	?	?	?	0.00	?
E	4.24	3.54	1.41	?	0.00

- Consider distance matrix of size 6X6 as shown below
- Beginning with 6 clusters as there are 6 objects
- Goal is to form single cluster
- In each iteration we find closest pair of cluster and merge it
- Group closest cluster D,F into cluster(D,F)
- Using single linkage we can specify the minimum distance between objects of the two clusters

Using the input distance matrix, distance between cluster (D, F) and cluster A is computed as

$$d_{(D,F)\to A} = \min\left(d_{DA}, d_{FA}\right) = \min\left(3.61, 3.20\right) = 3.20$$

Distance between cluster (D, F) and cluster B is

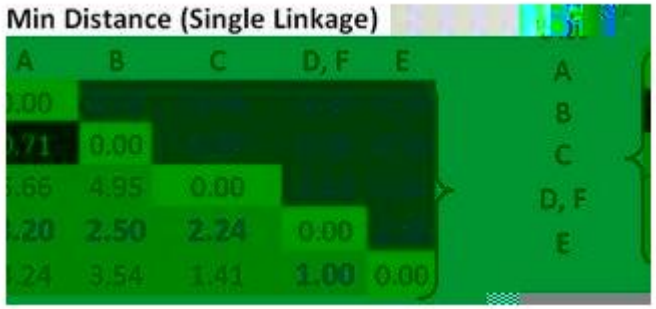
$$d_{(D,F)\to B} = \min\left(d_{DB}, d_{FB}\right) = \min\left(2.92, 2.50\right) = 2.50$$

Similarly, distance between cluster (D, F) and cluster C is

$$d_{(D,F)\to C} = \min\left(d_{DC}, d_{FC}\right) = \min\left(2.24, 2.50\right) = 2.24$$

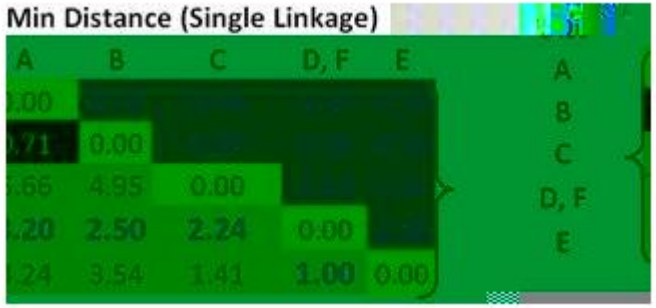
Finally, distance between cluster E and cluster (D, F) is calculated as

$$d_{E\to(D,F)} = \min\left(d_{ED}, d_{EF}\right) = \min\left(1.00, 1.12\right) = 1.00$$



Looking at the lower triangular updated distance matrix, we found out that the closest distance between cluster B and cluster A is now 0.71. Thus, we group cluster A and cluster B into a single cluster name (A, B).

Now we update the distance matrix. Aside from the first row and first column, all the other elements of the new distance matrix are not changed.



Dist	A,B	C	(D, F)	E
A,B	0	?	?	?
C	?	0	2.24	1.41
(D, F)	?	2.24	0	1.00
E	?	1.41	1.00	0

Using the input distance matrix (size 6 by 6), distance between cluster C and cluster (D, F) is computed as $d_{C\to(A,B)} = \min\left(d_{CA}, d_{CB}\right) = \min\left(5.66, 4.95\right) = 4.95$

Distance between cluster (D, F) and cluster (A, B) is the minimum distance between all objects involves in the two clusters

$$d_{(D,F)\to(A,B)} = \min\left(d_{DA}, d_{DB}, d_{FA}, d_{FB}\right) = \min\left(3.61, 2.92, 3.20, 2.50\right) = 2.50$$

Similarly, distance between cluster E and (A, B) is

$$d_{E\to(A,B)} = \min\left(d_{EA}, d_{EB}\right) = \min\left(4.24, 3.54\right) = 3.54$$

Then the updated distance matrix is

Min Distance (Single Linkage)

Dist	A,B	C	(D, F)	E
A,B	0	4.95	2.50	3.54
C	**4.95**	0	2.24	1.41
(D, F)	**2.50**	2.24	0	1.00
E	**3.54**	1.41	1.00	0

Observing the lower triangular of the updated distance matrix, we can see that the closest distance between clusters happens between cluster E and (D, F) at distance 1.00. Thus, we cluster them together into cluster ((D, F), E).

The updated distance matrix is given below.

Min Distance (Single Linkage)

Dist	(A,B)	C	(D, F), E
(A,B)	0.00	4.95	2.50
C	4.95	0.00	1.41
(D, F), E	**2.50**	**1.41**	0.00

Distance between cluster ((D, F), E) and cluster (A, B) is calculated as

$$d_{((D,F),E)\to(A,B)} = \min\left(d_{DA}, d_{DB}, d_{FA}, d_{FB}, d_{EA}, d_{EB}\right) = \min\left(3.61, 2.92, 3.20, 2.50, 4.24, 3.54\right) = 2.50$$

Distance between cluster ((D, F), E) and cluster C yields the minimum distance of 1.41. This distance is computed as $d_{((D,F),E)\to C} = \min\left(d_{DC}, d_{FC}, d_{EC}\right) = \min\left(2.24, 2.50, 1.41\right) = 1.41$. After that, we merge cluster ((D, F), E) and cluster C into a new cluster name (((D, F), E), C).

Min Distance (Single Linkage)

Dist	(A,B)	(D, F), E),C
(A,B)	0.00	2.50
((D, F), E),C	**2.50**	0.00

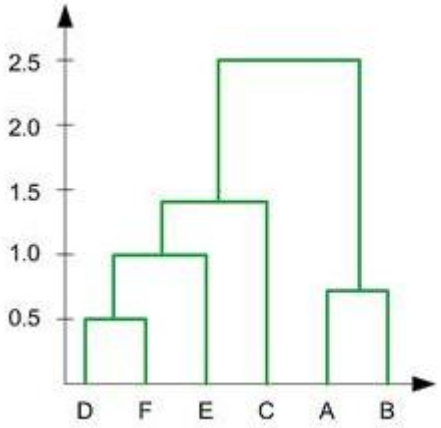
The minimum distance of 2.5 is the result of the following computation

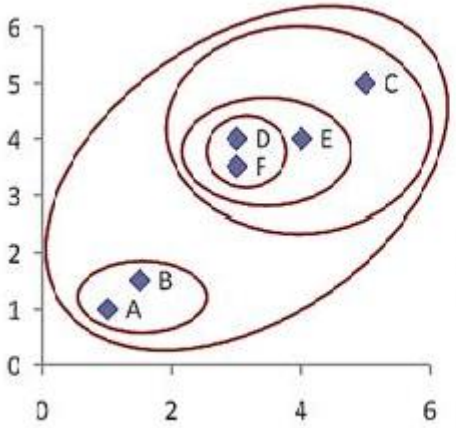
$$d_{(((D,F),E),C)\to(A,B)} = \min\left(d_{DA}, d_{DB}, d_{FA}, d_{FB}, d_{EA}, d_{EB}, d_{CA}, d_{CB}\right)$$

$$d_{(((D,F),E),C)\to(A,B)} = \min\left(3.61, 2.92, 3.20, 2.50, 4.24, 3.54, 5.66, 4.95\right) = 2.50$$

Now if we merge the remaining two clusters, we will get only single cluster contain the whole 6 objects. Thus, our computation is finished. We summarized the results of computation as follow:

1. In the beginning we have 6 clusters: A, B, C, D, E and F
2. We merge cluster D and F into cluster (D, F) at distance **0.50**
3. We merge cluster A and cluster B into (A, B) at distance **0.71**
4. We merge cluster E and (D, F) into ((D, F), E) at distance **1.00**
5. We merge cluster ((D, F), E) and C into (((D, F), E), C) at distance **1.41**
6. We merge cluster (((D, F), E), C) and (A, B) into ((((D, F), E), C), (A, B)) at distance **2.50**
7. The last cluster contain all the objects, thus conclude the computation





	X1	X2
A	1	1
B	1.5	1.5
C	5	5
D	3	4
E	4	4
F	3	3.5

Quality: What Is Good Clustering?

- A good clustering method will produce high quality clusters
  - high intra-class similarity: cohesive within clusters
  - low inter-class similarity: distinctive between clusters
- The quality of a clustering method depends on
  - the similarity measure used by the method
  - its implementation, and
  - Its ability to discover some or all of the hidden patterns

Measure the Quality of Clustering

- Dissimilarity/Similarity metric
  - Similarity is expressed in terms of a distance function, typically metric: $d(i, j)$
  - The definitions of distance functions are usually rather different for interval-scaled, boolean, categorical, ordinal ratio, and vector variables
  - Weights should be associated with different variables based on applications and data semantics
- Quality of clustering:
  - There is usually a separate "quality" function that measures the "goodness" of a cluster.
  - It is hard to define "similar enough" or "good enough"
    - The answer is typically highly subjective

Considerations for Cluster Analysis

- Partitioning criteria
  - Single level vs. hierarchical partitioning (often, multi-level hierarchical partitioning is desirable)
- Separation of clusters
  - Exclusive (e.g., one customer belongs to only one region) vs. non-exclusive (e.g., one document may belong to more than one class)
- Similarity measure
  - Distance-based (e.g., Euclidian, road network, vector) vs. connectivity-based (e.g., density or contiguity)
- Clustering space
  - Full space (often when low dimensional) vs. subspaces (often in high-dimensional clustering)

Requirements and Challenges

- Scalability
  - Clustering all the data instead of only on samples
- Ability to deal with different types of attributes
  - Numerical, binary, categorical, ordinal, linked, and mixture of these
- Constraint-based clustering
  - User may give inputs on constraints
  - Use domain knowledge to determine input parameters
- Interpretability and usability
- Others
  - Discovery of clusters with arbitrary shape
  - Ability to deal with noisy data
  - Incremental clustering and insensitivity to input order
  - High dimensionality

Extensions to Hierarchical Clustering

- Major weakness of agglomerative clustering methods
  - Can never undo what was done previously
  - Do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects
- **CURE(Clustering Using Representatives)**
- Integration of hierarchical & distance-based clustering
  - BIRCH (1996): uses CF-tree and incrementally adjusts the quality of sub-clusters
  - CHAMELEON (1999): hierarchical clustering using dynamic modeling

CURE(Clustering Using Representatives)

- It is a hierarchical based clustering technique, that adopts a middle ground between the centroid based and the all-point extremes.
- Hierarchical clustering is a type of clustering, that starts with a single point cluster, and moves to merge with another cluster, until the desired number of clusters are formed.
- It is used for identifying the spherical and non-spherical clusters.
- It is useful for discovering groups and identifying interesting distributions in the underlying data.
- Instead of using one point centroid, as in most of data mining algorithms, CURE uses a set of well-defined representative points, for efficiently handling the clusters and eliminating the outliers.

Algorithm:

- Draw a random sample.
- Partition the random sample.
- Partially cluster the partition.
- Outliers are identified and eliminated.
- The partial clusters obtained are clubbed into clustered.
- Label the result on storage.

BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies)

- Zhang, Ramakrishnan & Livny, SIGMOD'96
- Incrementally construct a CF (Clustering Feature) tree, a hierarchical data structure for multiphase clustering
  - Phase 1: scan DB to build an initial in-memory CF tree (a multi-level compression of the data that tries to preserve the inherent clustering structure of the data)
  - Phase 2: use an arbitrary clustering algorithm to cluster the leaf nodes of the CF-tree
- *Scales linearly*: finds a good clustering with a single scan and improves the quality with a few additional scans
- *Weakness:* handles only numeric data, and sensitive to the order of the data record

Clustering Feature Vector in BIRCH

Clustering Feature (CF): *CF = (N, LS, SS)*

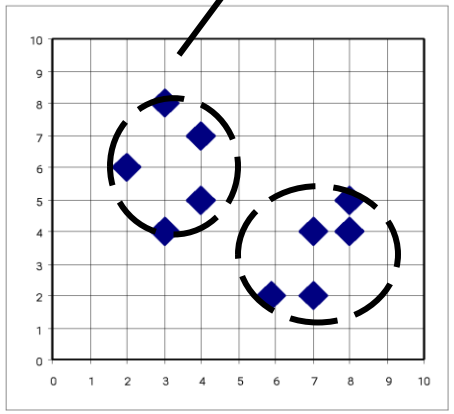
N: **Number of data points**

LS: linear sum of N points: $\sum_{i=1}^{N} X_i$

SS: square sum of N points

$\sum_{i=1}^{N} X_i^2$

CF = (5, (16,30),(54,190))

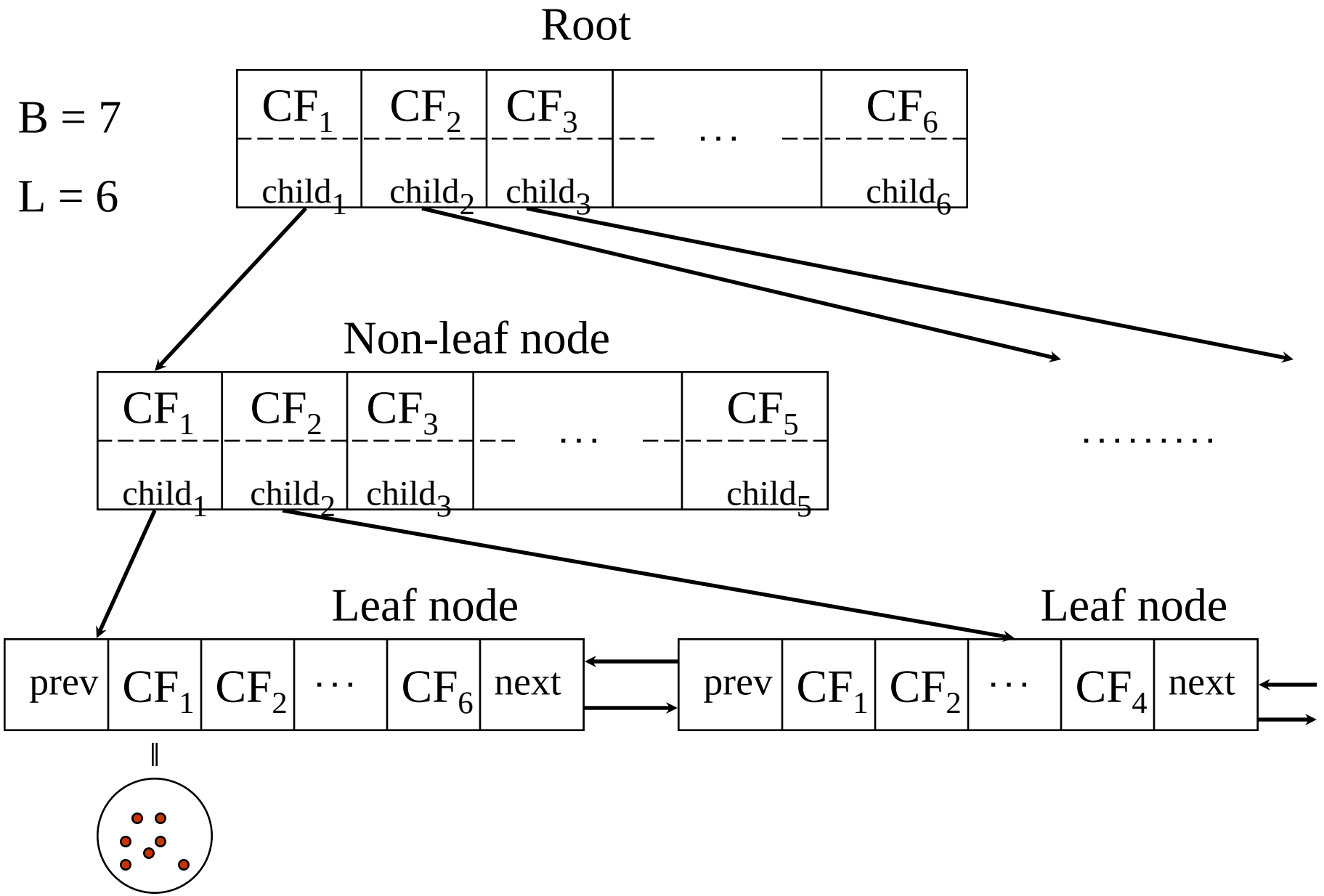


(3,4)
(2,6)
(4,5)
(4,7)
(3,8)

CF-Tree in BIRCH

- Clustering feature:
  - Summary of the statistics for a given subcluster: the 0-th, 1st, and 2nd moments of the subcluster from the statistical point of view
  - Registers crucial measurements for computing cluster and utilizes storage efficiently
- A CF tree is a height-balanced tree that stores the clustering features for a hierarchical clustering
  - A nonleaf node in a tree has descendants or “children”
  - The nonleaf nodes store sums of the CFs of their children
- A CF tree has two parameters
  - Branching factor: max # of children
  - Threshold: max diameter of sub-clusters stored at the leaf nodes

The CF Tree Structure



The Birch Algorithm

- Cluster Diameter $\sqrt{\frac{1}{n(n-1)}\sum(x_i - x_j)^2}$

- For each point in the input
  - Find closest leaf entry
  - Add point to leaf entry and update CF
  - If entry diameter > max_diameter, then split leaf, and possibly parents
- Algorithm is O(n)
- Concerns
  - Sensitive to insertion order of data points
  - Since we fix the size of leaf nodes, so clusters may not be so natural
  - Clusters tend to be spherical given the radius and diameter measures

Fuzzy Set and Fuzzy Cluster

- Clustering methods discussed so far
  - Every data object is assigned to exactly one cluster
- Some applications may need for fuzzy or soft cluster assignment
  - Ex. An e-game could belong to both entertainment and software
- Methods: fuzzy clusters and probabilistic model-based clusters
- Fuzzy cluster: A fuzzy set *S:* $F_S : X \rightarrow [0, 1]$ (value between 0 and 1)
- Example: Popularity of cameras is defined as a fuzzy mapping

Camera	Sales (units)
A	50
B	1320
C	860
D	270

$$\text{Pop}(o) = \begin{cases} 1 & \text{if } 1,000 \text{ or more units of } o \text{ are sold} \\ \frac{i}{1000} & \text{if } i \ (i < 1000) \text{ units of } o \text{ are sold} \end{cases}$$

- Then, *A*(0.05), *B*(1), *C*(0.86), *D*(0.27)

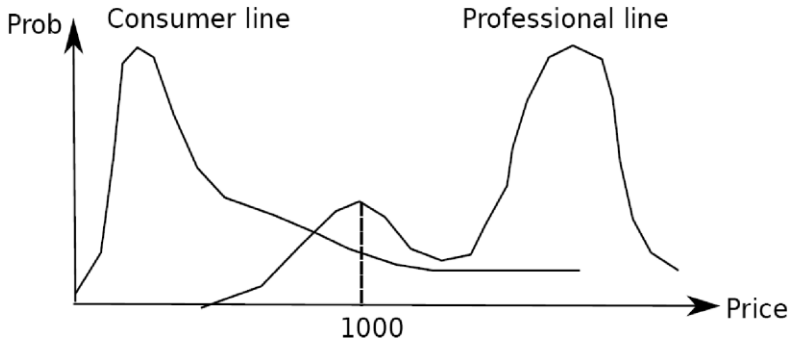
Advantages and Disadvantages of Mixture Models

- Strength
  - Mixture models are more general than partitioning and fuzzy clustering
  - Clusters can be characterized by a small number of parameters
  - The results may satisfy the statistical assumptions of the generative models
- Weakness
  - Converge to local optimal (overcome: run multi-times w. random initialization)
  - Computationally expensive if the number of distributions is large, or the data set contains very few observed data points
  - Need large data sets
  - Hard to estimate the number of clusters

Probabilistic Model-Based Clustering

- Cluster analysis is to find hidden categories.
- A hidden category (i.e., *probabilistic cluster)* is a distribution over the data space, which can be mathematically represented using a probability density function (or distribution function).

- Ex. 2 categories for digital cameras sold
  - consumer line vs. professional line
  - density functions f_1, f_2 for C_1, C_2
  - obtained by probabilistic clustering



- A **mixture model** assumes that a set of observed objects is a mixture of instances from multiple probabilistic clusters, and conceptually each observed object is generated independently
- **Out task**: infer a set of *k* probabilistic clusters that is mostly likely to generate *D* using the above data generation process

Model-Based Clustering

- A set C of k probabilistic clusters $C_1, \ldots, C_k$ with probability density functions $f_1, \ldots, f_k$, respectively, and their probabilities $\omega_1, \ldots, \omega_k$.
- Probability of an object o generated by cluster C_j is $P(o|C_j) = \omega_j f_j(o)$
- Probability of o generated by the set of cluster $\boldsymbol{C}$ is $P(o|\boldsymbol{C}) = \sum_{j=1}^{k} \omega_j f_j(o)$

- Since objects are assumed to be generated independently, for a data set D = {o_1, …, o_n}, we have,

$$P(D|\boldsymbol{C}) = \prod_{i=1}^{n} P(o_i|\boldsymbol{C}) = \prod_{i=1}^{n} \sum_{j=1}^{k} \omega_j f_j(o_i)$$

- Task: Find a set C of k probabilistic clusters s.t. $P(D|\boldsymbol{C})$ is maximized
- However, maximizing $P(D|\boldsymbol{C})$ is often intractable since the probability density function of a cluster can take an arbitrarily complicated form
- To make it computationally feasible (as a compromise), assume the probability density functions being some parameterized distributions

Univariate Gaussian Mixture Model

- $O = \{o_1, \ldots, o_n\}$ (n observed objects), $\Theta = \{\theta_1, \ldots, \theta_k\}$ (parameters of the k distributions), and $P_j(o_i | \theta_j)$ is the probability that o_i is generated from the j-th distribution using parameter θ_i, we have

$$P(o_i|\Theta) = \sum_{j=1}^{k} \omega_j P_j(o_i|\Theta_j) \qquad P(\mathbf{O}|\Theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} \omega_j P_j(o_i|\Theta_j)$$

- Univariate Gaussian mixture model
  - Assume the probability density function of each cluster follows a 1-d Gaussian distribution. Suppose that there are k clusters.
  - The probability density function of each cluster are centered at μ_j with standard deviation σ_j, θ_j, = (μ_j, σ_j), we have

$$P(o_i|\Theta_j) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma^2}} \qquad P(o_i|\Theta) = \sum_{j=1}^{k} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma^2}}$$

$$P(\mathbf{O}|\Theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma^2}}$$

The EM (Expectation Maximization) Algorithm

- The k-means algorithm has two steps at each iteration:
  - **Expectation Step** (E-step): Given the current cluster centers, each object is assigned to the cluster whose center is closest to the object: An object is *expected to belong to the closest cluster*
  - **Maximization Step** (M-step): Given the cluster assignment, for each cluster, the algorithm *adjusts the center* so that *the sum of distance* from the objects assigned to this cluster and the new center is minimized
- **The (EM) algorithm:** A framework to approach maximum likelihood or maximum a posteriori estimates of parameters in statistical models.
  - **E-step** assigns objects to clusters according to the current fuzzy clustering or parameters of probabilistic clusters
  - **M-step** finds the new clustering or parameters that maximize the sum of squared error (SSE) or the expected likelihood

Gaussian Mixture Model

Gaussian Mixture Model

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

A Gaussian mixture model (GMM) is a category of probabilistic model that states that all generated data points are derived from a mixture of finite Gaussian distributions that have no known parameters

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Cntd..

In one dimension the probability density function of a Gaussian Distribution is given by

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

Multivariate Gaussian

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi|\Sigma|)^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

mean

covariance

Gaussian Mixture

- Linear combination of Gaussian

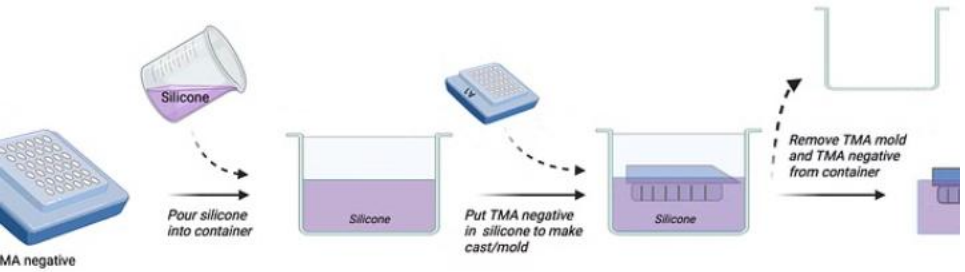
$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \quad \text{where} \quad \sum_{k=1}^{K} \pi_k = 1, \quad 0 \leqslant \pi_k \leqslant 1$$

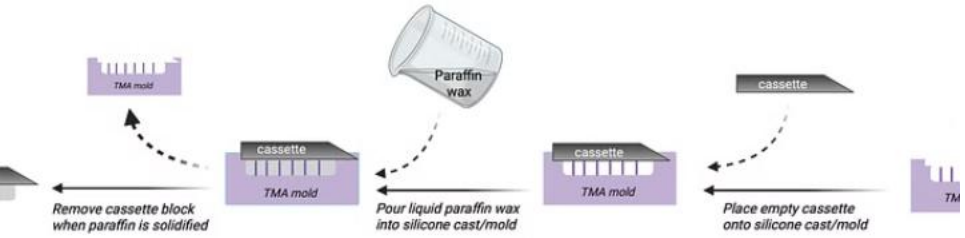
parameters to be estimated

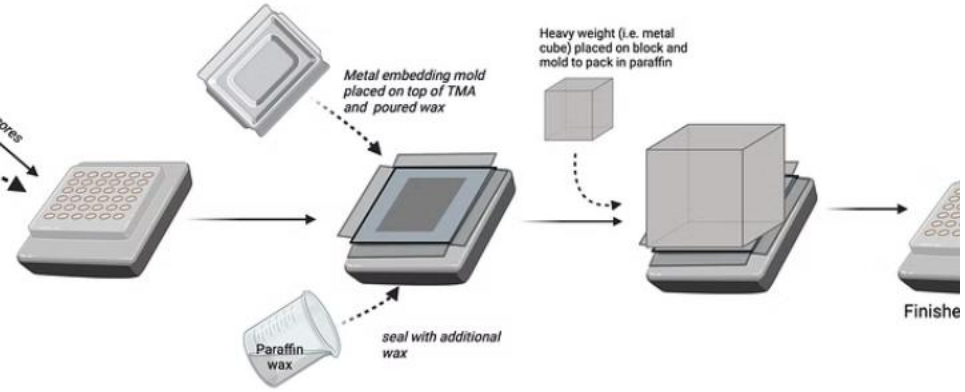
Human Tumor Microarray Data

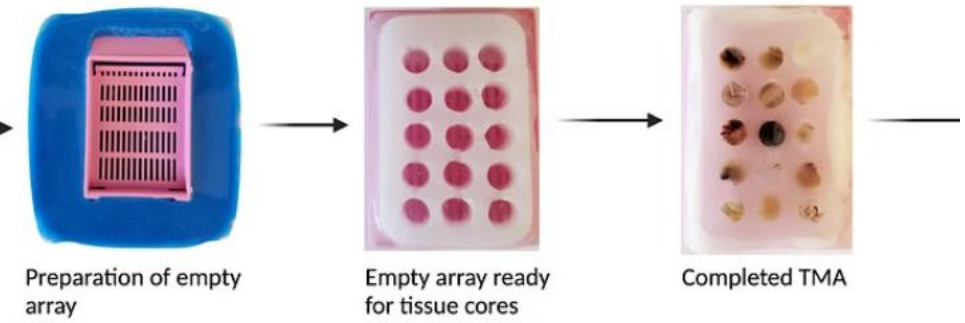
- The human tumor microarray, is a cutting-edge technology that revolutionizes cancer research and diagnosis.

- This innovative approach harnesses the potential of machine learning to uncover invaluable insights into the complex tumor microenvironment.







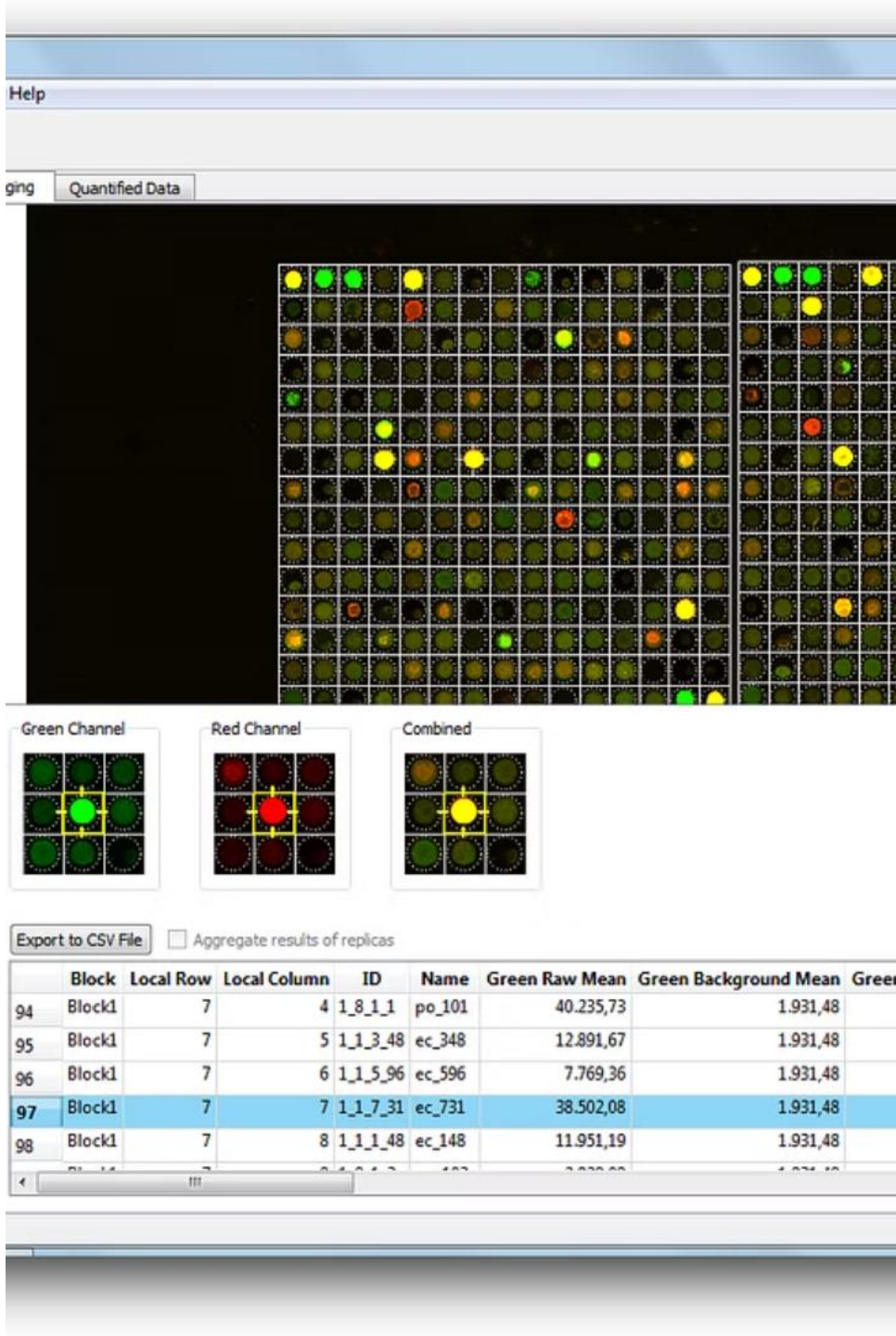


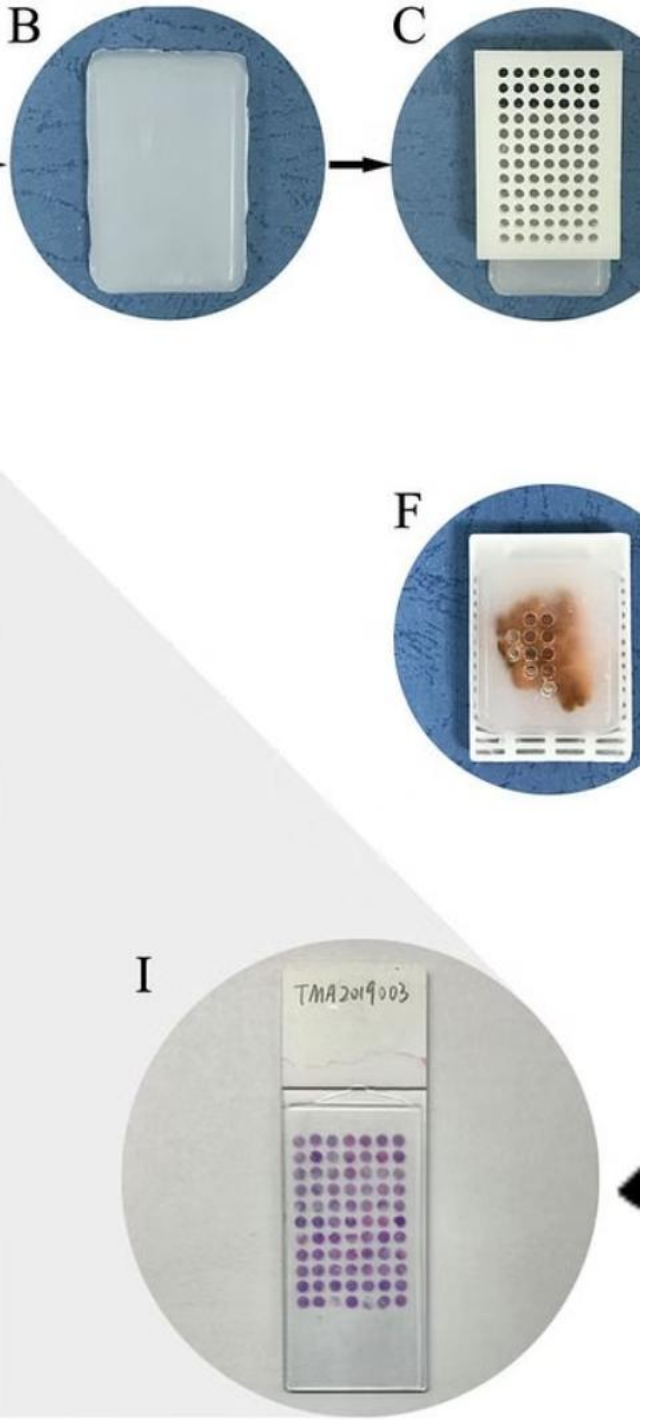
Overview of Tissue Microarray Technology

Tissue microarray (TMA) is an innovative technology that allows for high-throughput analysis of tissue samples. In a TMA, hundreds of tiny tissue cores are precisely arrayed on a single paraffin block, enabling **simultaneous immunohistochemical or in situ hybridization analysis** of multiple patient samples on a single slide.

Role of Machine Learning in Tumor Microarray Analysis

Machine learning algorithms have become invaluable tools for analyzing the vast amounts of data generated by tumor microarray technologies. These powerful techniques can uncover complex patterns and relationships within the data, enabling researchers to better understand tumor biology and develop more targeted therapies.





Algorithms for Tumor Microarray Analysis

1

Image Segmentation

Applying image segmentation algorithms to isolate individual tissue cores and separate them from the background.

2

Feature Extraction

Extracting quantitative features from the tissue cores, such as staining intensity, texture, and morphological properties.

3

Supervised Classification

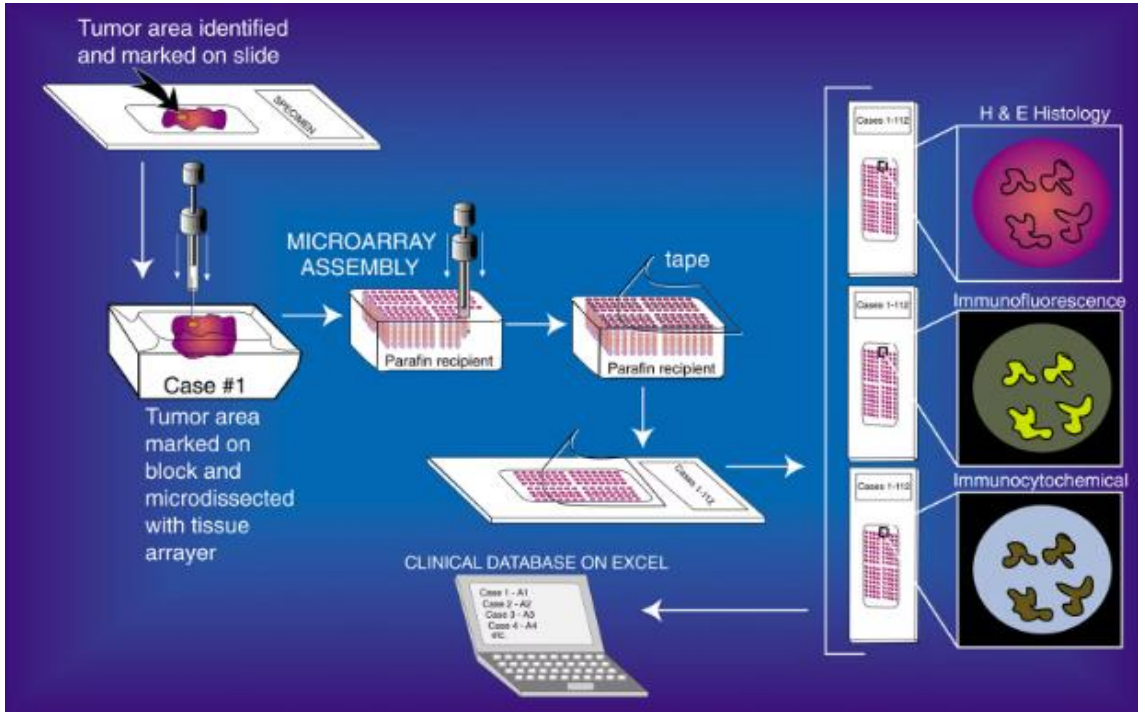
Using machine learning algorithms like decision trees, random forests, and support vector machines to classify tumor samples into different subtypes or grades.

Image Processing and Feature Extraction

Image processing plays a crucial role in analyzing tissue microarray data. Key steps include image segmentation, cell detection, and feature extraction.

Sophisticated algorithms extract quantitative measurements from the images, such as cell counts, protein expression levels, and spatial distributions.

These image-derived features serve as inputs to machine learning models for tumor classification and subtyping.



Supervised Learning Techniques for Tumor Classification

1

Logistic Regression

Predicts tumor type based on features

2

Support Vector Machines

Identifies optimal hyperplane to separate tumor classes

3

Random Forests

Ensemble of decision trees for robust classification

Supervised learning algorithms are powerful tools for classifying tumor types based on the microarray data. Logistic regression, support vector machines, and random forests are commonly used techniques that can accurately predict tumor subtypes by learning from labeled training data. These models excel at identifying complex patterns in the high-dimensional tumor profiles.

Unsupervised Clustering Algorithms for Tumor Subtyping

Unsupervised clustering algorithms play a crucial role in identifying distinct tumor subtypes within a heterogeneous tumor microarray dataset. These algorithms can uncover hidden patterns and groupings without relying on pre-defined labels or classes.

K-Means Clustering	Groups tumors into k distinct clusters based on similarity of gene expression profiles. Useful for identifying broad tumor subtypes.
Hierarchical Clustering	Builds a hierarchy of clusters, allowing visualization of relationships between tumor samples. Helps identify subtypes and their hierarchical structure.
Gaussian Mixture Models	Assumes data is generated from a mixture of Gaussian distributions. Can identify overlapping tumor subtypes with probabilistic assignments.

These algorithms uncover hidden structures in the tumor microarray data, enabling researchers to identify novel tumor subtypes that may have distinct molecular profiles and clinical outcomes. The discovered subtypes can then be further studied and validated using supervised learning techniques.

Applications of Tissue Microarray in Cancer Research

Biomarker Discovery

Tissue microarrays enable rapid screening of tissue samples for potential biomarkers associated with cancer prognosis and treatment response.

Drug Development

Tissue microarrays allow efficient evaluation of drug candidates and their impact on various tumor types, accelerating the drug discovery process.

High-Throughput Analysis

The compact array format enables comprehensive, high-throughput analysis of large cohorts of patient samples, providing valuable insights

Pathology Validation

Tissue microarrays facilitate the validation of novel immunohistochemical markers and their association with clinicopathological features of cancer.

Advantages of Tissue Microarray

1

High-Throughput Analysis

Tissue microarray enables the simultaneous analysis of hundreds of tissue samples on a single slide, dramatically increasing the efficiency of cancer research.

2

Reduced Tissue Consumption

The miniaturized format of tissue microarray requires only small amounts of tissue, preserving precious patient samples for multiple analyses.

3

Standardized Conditions

Tissue microarray ensures consistent staining and analysis conditions across all samples, improving the reliability and reproducibility of results.

4

Cost-Effectiveness

The streamlined workflow and reduced tissue requirements of tissue microarray make it a cost-effective tool for cancer research and biomarker discovery.

Disadvantages of Tissue Microarray

1 Limited Representativeness

Tissue microarrays typically contain small tissue cores from different regions of a tumour or from multiple tumours.

2 Tissue Damage and Degradation

The process of constructing tissue microarrays involves punching small cores from paraffin-embedded tissue blocks.

3 Sampling Bias

The selection of tissue cores for inclusion in a microarray may introduce sampling bias, particularly if cores are selected based on subjective criteria or if certain tissue regions are overrepresented or underrepresented.

4 Loss of Morphological Context

Tissue microarrays sacrifice the spatial context of individual tissue samples, as multiple cores are arranged in a grid-like pattern on a single slide.

Vector Quantization

Vectors are mathematical entities that represent quantities that have both magnitude and direction. In vector quantization, these vectors typically represent data points in a multi-dimensional space. For example, in image compression, each pixel's color values can be represented as a vector in a high-dimensional color space.

Vector quantization is a technique used in data compression. It involves the process of representing a large set of data points with a smaller set of representative points called centroids or codewords.

Vector Quantization : Basic Working

- Codebook Generation: A codebook is created initially, containing a predefined number of codewords or centroids. These codewords can be randomly selected or generated based on some criteria.
- Quantization: Each input vector in the dataset is then assigned to the nearest codeword in the codebook. This process is called quantization, where each vector is mapped to a single codeword.
- Compression: Instead of storing the entire dataset, only the indices of the codewords to which each vector is assigned are stored. This results in significant compression, especially when the dataset is large.
- Reconstruction: To reconstruct the original data, the indices stored along with the codebook are used to retrieve the corresponding codewords, which are then used to approximate the original vectors.

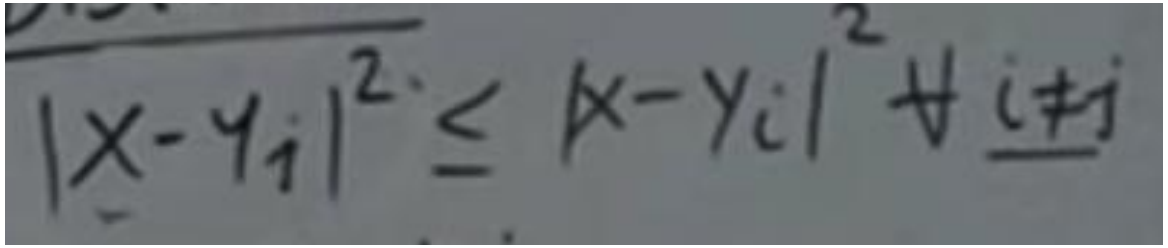
Vector Quantization : Basic Working

Example : {0.75 , 1.27} , {1.78 , 2.11}

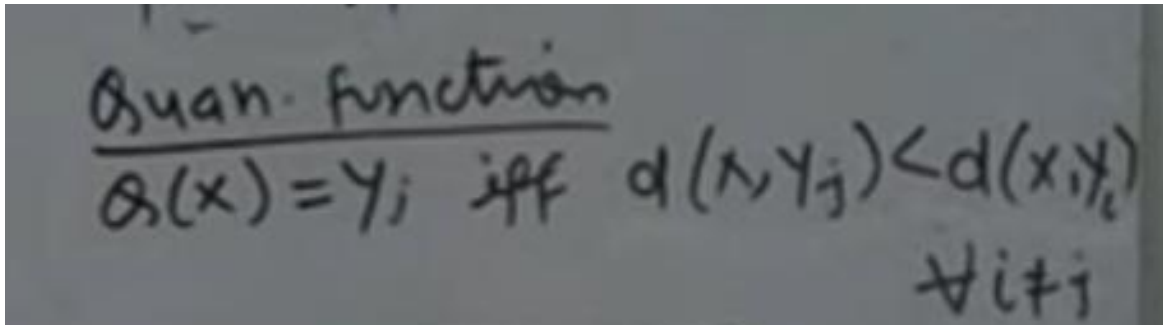
CODEBOOK

-2	-2
-1	-1
1	1
2	2

Distortion



Quantization Function



Source : Youtube (Rudra Singh - https://www.youtube.com/watch?v=qzhIXfxMDpM&t=191s)

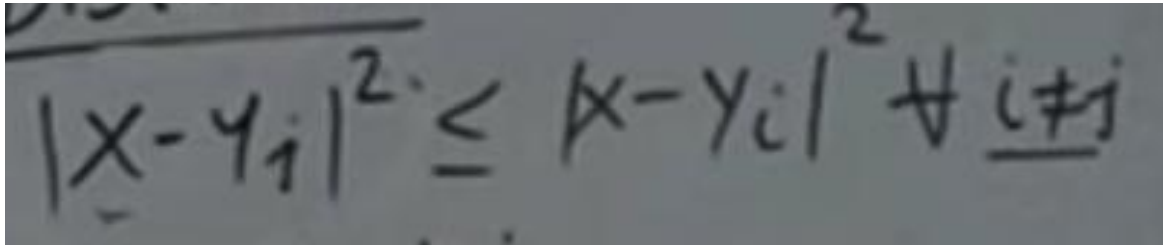
Vector Quantization : Basic Working

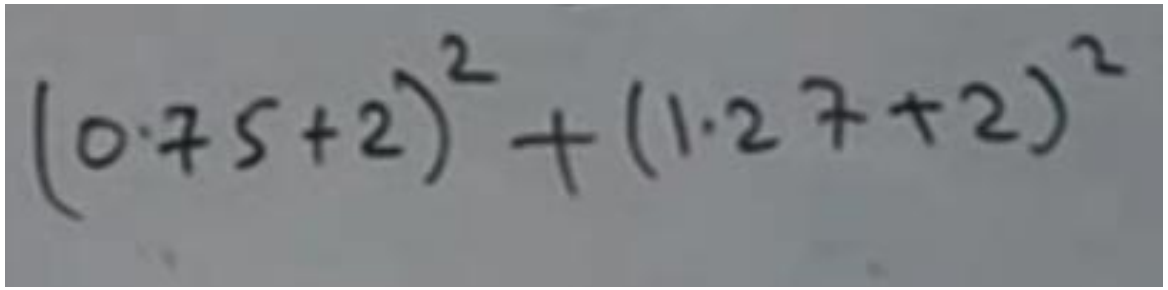
Example : {0.75 , 1.27} , {1.78 , 2.11}

CODEBOOK

-2	-2
-1	-1
1	1
2	2

Distortion





Source : Youtube (Rudra Singh - https://www.youtube.com/watch?v=qzhIXfxMDpM&t=191s)

Applications

Data Compression

Vector quantization is widely used in data compression applications such as image, audio, and video compression.

In image compression, for instance, vector quantization techniques like the Linde-Buzo-Gray (LBG) algorithm are employed to represent image blocks with a smaller set of representative codewords, achieving high compression ratios while preserving image quality.

Pattern Recognition

Vector quantization is utilized in pattern recognition tasks such as classification, clustering, and feature extraction.

In classification tasks, codewords obtained through vector quantization serve as informative features for training classifiers to recognize and classify different patterns or objects.

Data Analysis and Mining

Vector quantization is applied in data analysis and mining tasks such as clustering and anomaly detection.

In clustering tasks, vector quantization algorithms are used to group similar data points together based on their feature representations.

Advantages

- Data compression: Vector Quantization can achieve significant data compression with minimal loss of information, making it suitable for applications like image and audio compression.

- Noise reduction: Vector Quantization can help reduce noise in the data by replacing individual data points with representative codebook vectors, leading to smoother and more robust representations.

- Pattern recognition: Vector Quantization can be used to identify patterns or structures in the data, which can be useful for tasks like classification, clustering, and feature extraction.

Disadvantages

Fixed Codebook Size: Most vector quantization techniques require a fixed-size codebook, which needs to be predefined before the quantization process. Adapting the codebook size to accommodate variations in the data distribution or achieving optimal compression performance can be challenging.

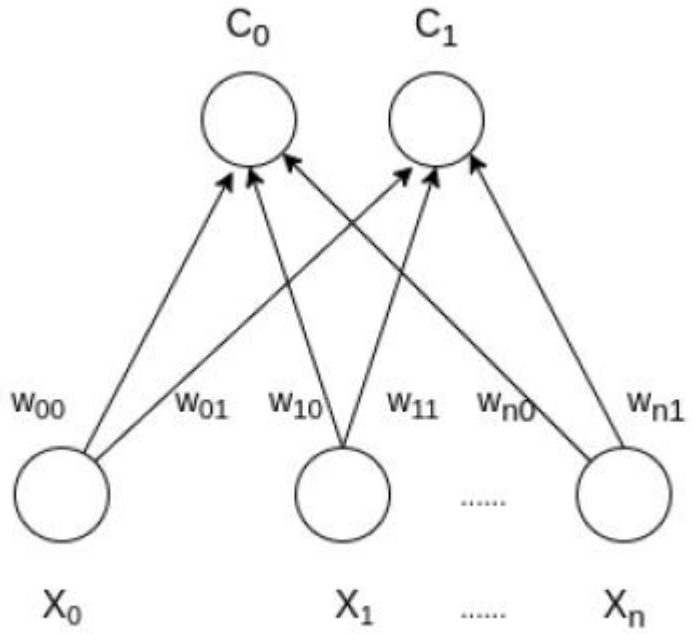
Limited Adaptability to Data Changes: Once the codebook is trained, it remains fixed and may not adapt well to changes in the input data distribution over time. This lack of adaptability may result in suboptimal quantization performance when the data distribution shifts or evolves.

High Computational Complexity: Depending on the size of the dataset and the dimensionality of the feature space, vector quantization algorithms can be computationally intensive, especially during the codebook training phase. This high computational complexity may limit the scalability of VQ algorithms to large datasets

Self Organizing Maps – Kohonen Maps

- **Self Organizing Map (or Kohonen Map or SOM)** is a type of Artificial Neural Network which is also inspired by biological models of neural systems from the 1970s.
- It follows an unsupervised learning approach and trained its network through a competitive learning algorithm.
- SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation.
- SOM has two layers, one is the Input layer and the other one is the Output layer.

- The architecture of the Self Organizing Map with two clusters and n input features of any sample is given below:



How do SOM works?

- Let's say an input data of size (m, n) where m is the number of training examples and n is the number of features in each example.
- First, it initializes the weights of size (n, C) where C is the number of clusters.
- Then iterating over the input data, for each training example, it updates the winning vector (weight vector with the shortest distance (e.g Euclidean distance) from training example). Weight updation rule is given by

$$w_{ij} = w_{ij}(old) + alpha(t) * (x_i^k - w_{ij}(old))$$

- where alpha is a learning rate at time t, j denotes the winning vector, i denotes the i^{th} feature of training example and k denotes the k^{th} training example from the input data.
- After training the SOM network, trained weights are used for clustering new examples. A new example falls in the cluster of winning vectors.

Algorithm

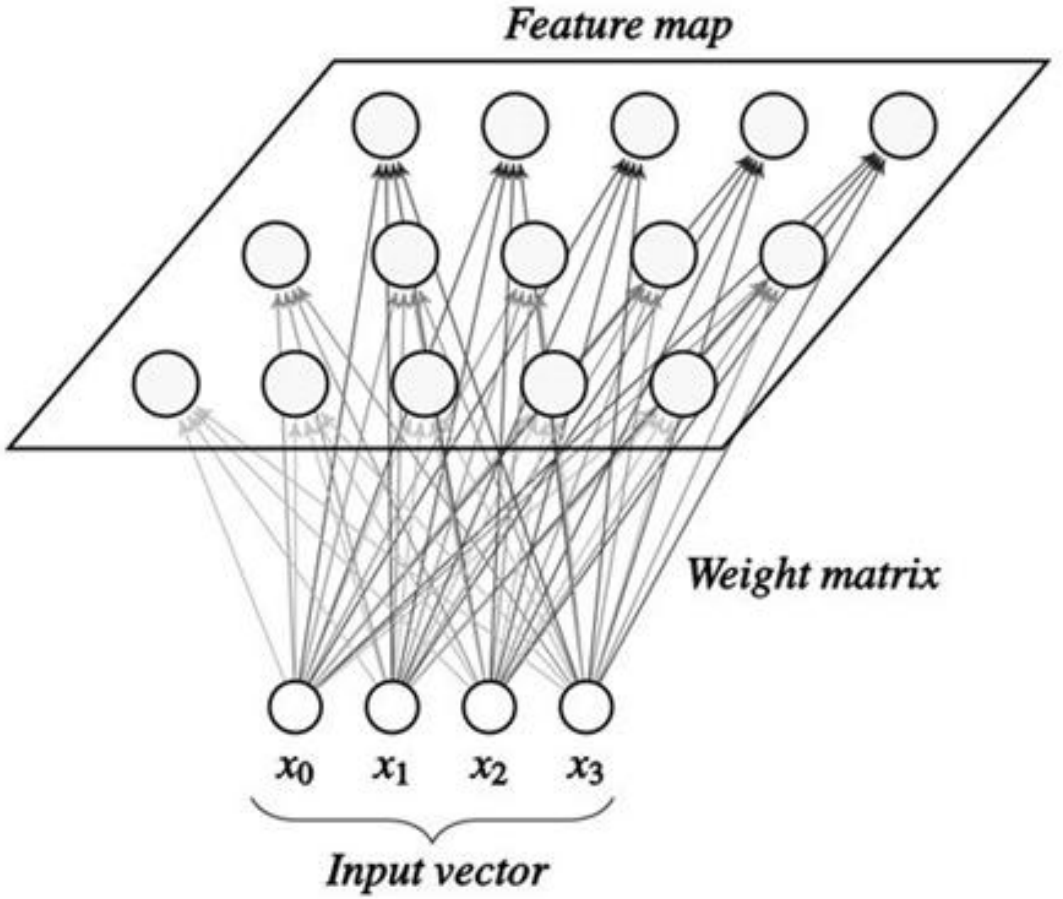
- **Training:**
- **Step 1:** Initialize the weights w_{ij} random value may be assumed. Initialize the learning rate α.
- **Step 2:** Calculate squared Euclidean distance.
- $D(j) = \Sigma (wij – xi)^2$ where i=1 to n and j=1 to m
- **Step 3:** Find index J, when D(j) is minimum that will be considered as winning index.
- **Step 4:** For each j within a specific neighborhood of j and for all i, calculate the new weight.
- $wij(new)=wij(old) + \alpha[xi – wij(old)]$
- **Step 5:** Update the learning rule by using :
- $\alpha(t+1) = 0.5 * t$
- **Step 6:** Test the Stopping Condition.

- **Output:**
  Test Sample s belongs to Cluster : 0
  Trained weights : [[0.6000000000000001, 0.8, 0.5, 0.9],
  [0.3333984375, 0.0666015625, 0.7, 0.3]]

Self Organization Maps

- The Self Organizing Map is one of the most popular neural models. It belongs to the category of the competitive learning network.
- The SOM is based on unsupervised learning, which means that is no human intervention is needed during the training and those little needs to be known about characterized by the input data.
- We could, for example, use the SOM for clustering membership of the input data. The SOM can be used to detect features inherent to the problem and thus has also been called SOFM the Self Origination Feature Map.

- The Self Organized Map was developed by Professor Kohenen and is used in many applications.
- the purpose of SOM is that it’s providing a data visualization technique that helps to understand high dimensional data by reducing the dimension of data to map.
- SOM also represents the clustering concept by grouping similar data.
- Therefore it can be said that the Self Organizing Map reduces data dimension and displays similarly among data.


Feature map
Weight matrix
x_0 x_1 x_2 x_3
Input vector

SOM

- A SOM does not need a target output to be specified, unlike many other types of networks.
- Instead, where the node weights match the input vector, that area of the lattice is selectively optimized to more closely resemble the data for the class the input vector is a member of.
- From an initial distribution of random weights, and over many iterations, the SOM eventually settles into a map of stable zones.
- Each zone is effectively a feature classifier, so you can think of the graphical output as a type of feature map of the input space.

SOM

Training occurs in several steps and over many iterations:

1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data and presented to the lattice.
3. Every node is examined to calculate which ones weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU).

.

SOM

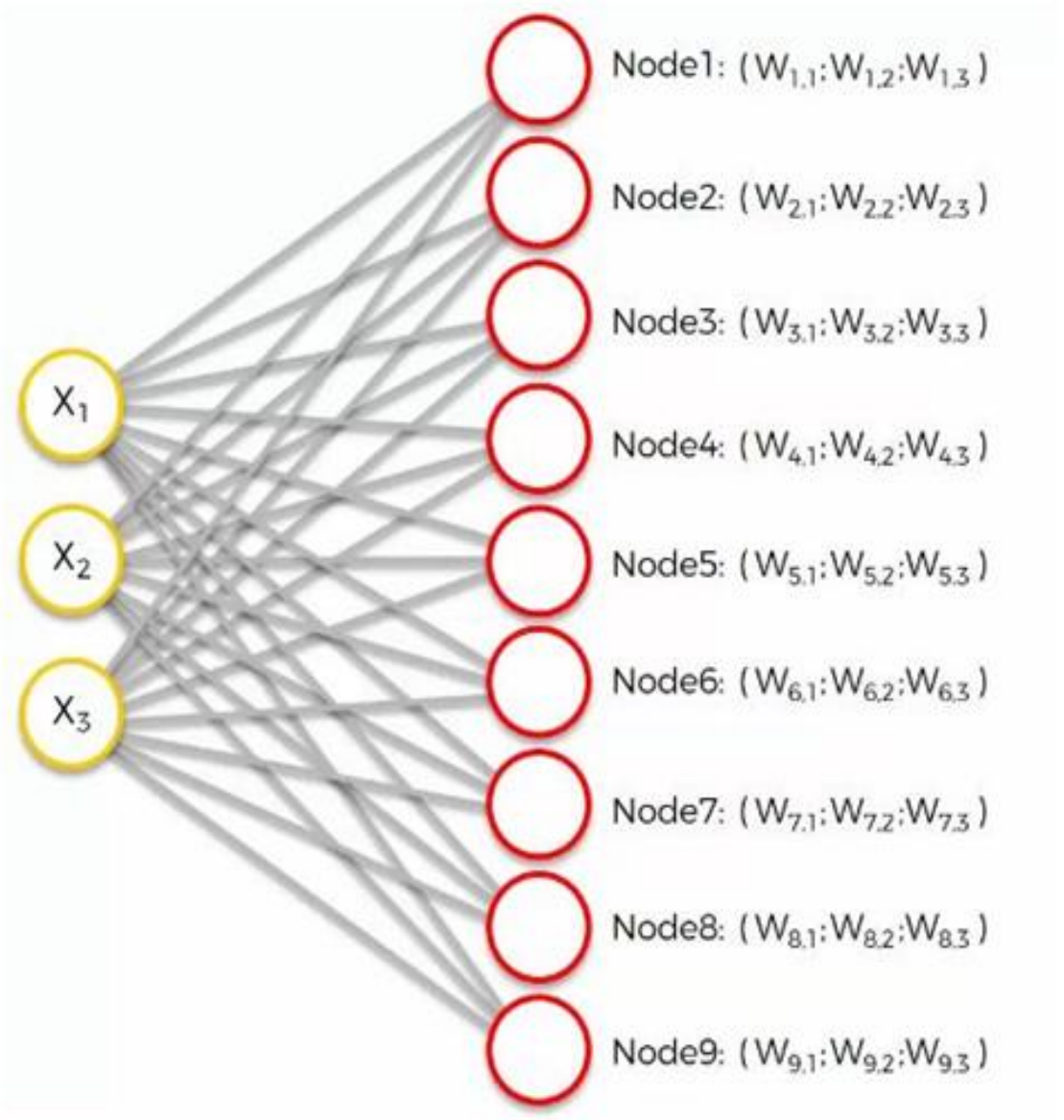
4. The radius of the neighborhood of the BMU is now calculated. This value starts large, typically set to the ‘radius’ of the lattice, but diminishes each time step. Any nodes within this radius are deemed inside the BMU’s neighborhood.
5. Each neighboring node’s (the nodes found in step 4) weights are adjusted to make them more like the input vector. The closer a node is to the BMU; the more its weights get altered.
6. Repeat step 2 for N iterations.

SOM

4. The radius of the neighborhood of the BMU is now calculated. This value starts large, typically set to the ‘radius’ of the lattice, but diminishes each time step. Any nodes within this radius are deemed inside the BMU’s neighborhood.
5. Each neighboring node’s (the nodes found in step 4) weights are adjusted to make them more like the input vector. The closer a node is to the BMU; the more its weights get altered.
6. Repeat step 2 for N iterations.

SOM

Example:


X1
X2
X3
Node1: (W1,1:W1,2:W1,3)
Node2: (W2,1:W2,2:W2,3)
Node3: (W3,1:W3,2:W3,3)
Node4: (W4,1:W4,2:W4,3)
Node5: (W5,1:W5,2:W5,3)
Node6: (W6,1:W6,2:W6,3)
Node7: (W7,1:W7,2:W7,3)
Node8: (W8,1:W8,2:W8,3)
Node9: (W9,1:W9,2:W9,3)

SOM

Randomly initialize the values of the weights (close to 0 but not 0).

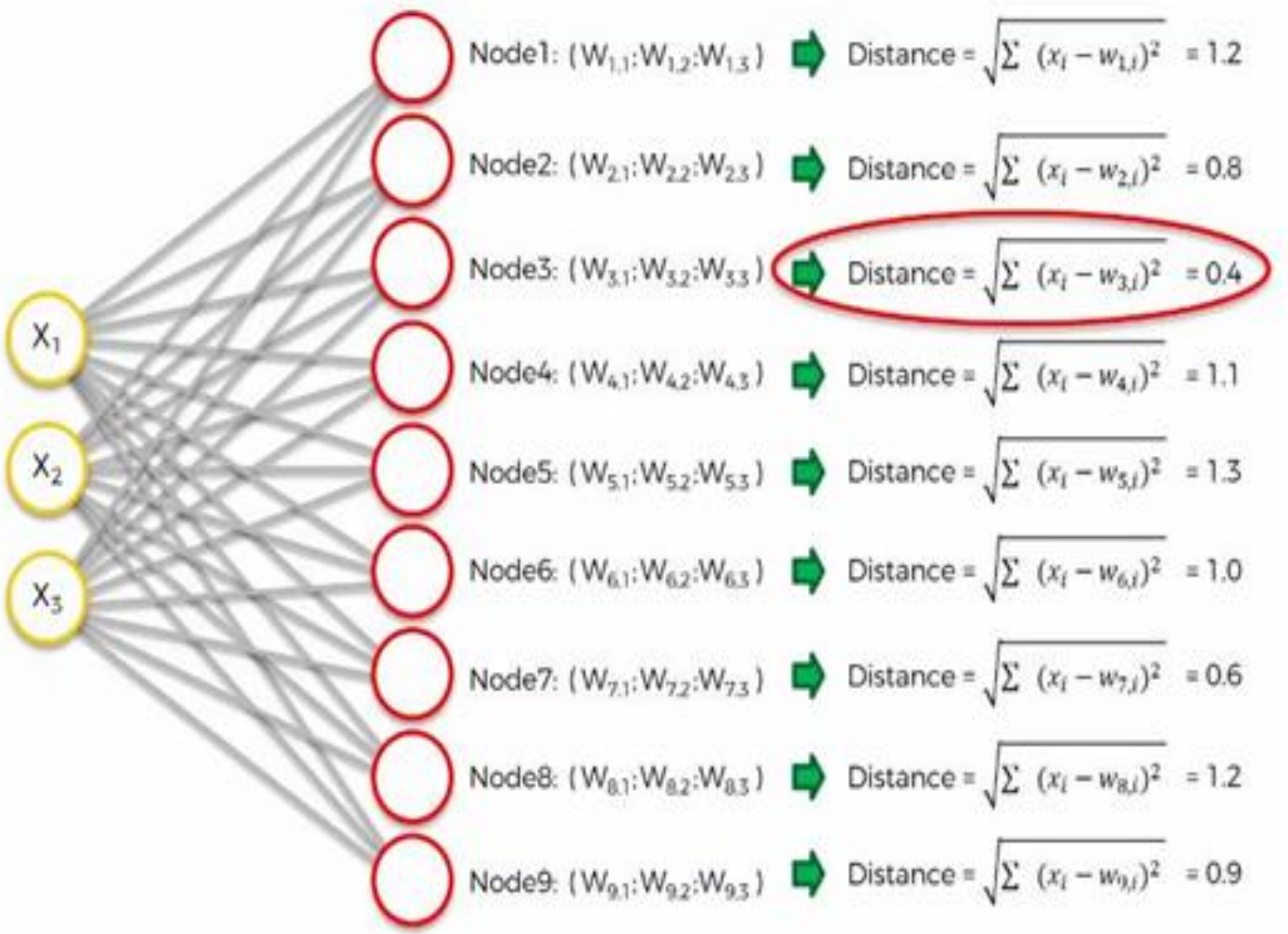
$W_{1,1} = 0.31$	$W_{1,2} = 0.22$	$W_{1,3} = 0.10$
$W_{2,1} = 0.21$	$W_{2,2} = 0.34$	$W_{2,3} = 0.19$
$W_{3,1} = 0.39$	$W_{3,2} = 0.42$	$W_{3,3} = 0.45$
$W_{4,1} = 0.25$	$W_{4,2} = 0.32$	$W_{4,3} = 0.62$
$W_{5,1} = 0.24$	$W_{5,2} = 0.31$	$W_{5,3} = 0.16$
$W_{6,1} = 0.52$	$W_{6,2} = 0.33$	$W_{6,3} = 0.42$
$W_{7,1} = 0.31$	$W_{7,2} = 0.22$	$W_{7,3} = 0.10$
$W_{8,1} = 0.12$	$W_{8,2} = 0.41$	$W_{8,3} = 0.19$
$W_{9,1} = 0.34$	$W_{9,2} = 0.40$	$W_{9,3} = 0.51$

Step 2: Calculating the Best Matching Unit with

$$\text{Distance} = \sqrt{\sum_{i=0}^{i=n}(X_i - W_i)^2}$$

SOM

Node number 3 is the closest with a distance of 0.4. We will call this node our BMU (best-matching unit).



SOM

Adjusting the Weights

Every node within the BMU's neighborhood (including the BMU) has its weight vector adjusted according to the following equation:

New Weights = Old Weights + Learning Rate (Input Vector — Old Weights)

$W(t+1) = W(t) + L(t) (V(t) — W(t))$

So according to our example Node 4 is the Best Match Unit (as you can see in step 2) corresponding to their weights:

SOM

$W_{3,1} = 0.39$ $W_{3,2} = 0.42$ $W_{3,3} = 0.45$

Input Vector: $X_1 = 0.7$ $X_2 = 0.6$ $X_3 = 0.9$

Learning rate = 0.5
So update that weight according to the above equation
For W3,1
New Weights = Old Weights + Learning Rate (Input Vector1 — Old Weights)
New Weights = 0.39 + 0.5 (0.7–0.39)
New Weights = 0.545
For W3,2
New Weights = Old Weights + Learning Rate (Input Vector2 — Old Weights)
New Weights = 0.42 + 0.5 (0.6–0.42)

SOM

New Weights = 0.51

For W3,3

New Weights = Old Weights + Learning Rate (Input Vector3 — Old Weights)

New Weights = 0.45 + 0.5 (0.9–0.45)

New Weights = 0.675

Updated weights

i

$W_{3,1} = 0.545$ $W_{3,2} = 0.51$ $W_{3,3} = 0.67$

Spectral clustering:

- Combining feature extraction and clustering (i.e., use the *spectrum* of the similarity matrix of the data to perform dimensionality reduction for clustering in fewer dimensions)
  - Normalized Cuts (Shi and Malik, CVPR'97 or PAMI'2000)
  - The Ng-Jordan-Weiss algorithm (NIPS'01)

Spectral Clustering: The Ng-Jordan-Weiss (NJW) Algorithm

- Given a set of objects o_1, …, o_n, and the distance between each pair of objects, dist(o_i, o_j), find the desired number k of clusters
- Calculate an affinity matrix W, where σ is a scaling parameter that controls how fast the affinity W_{ij} decreases as dist(o_i, o_j) increases. In NJW, set $W_{ij} = 0$

$$D_{ii} = \sum_{j=1}^{n} W_{ij}$$

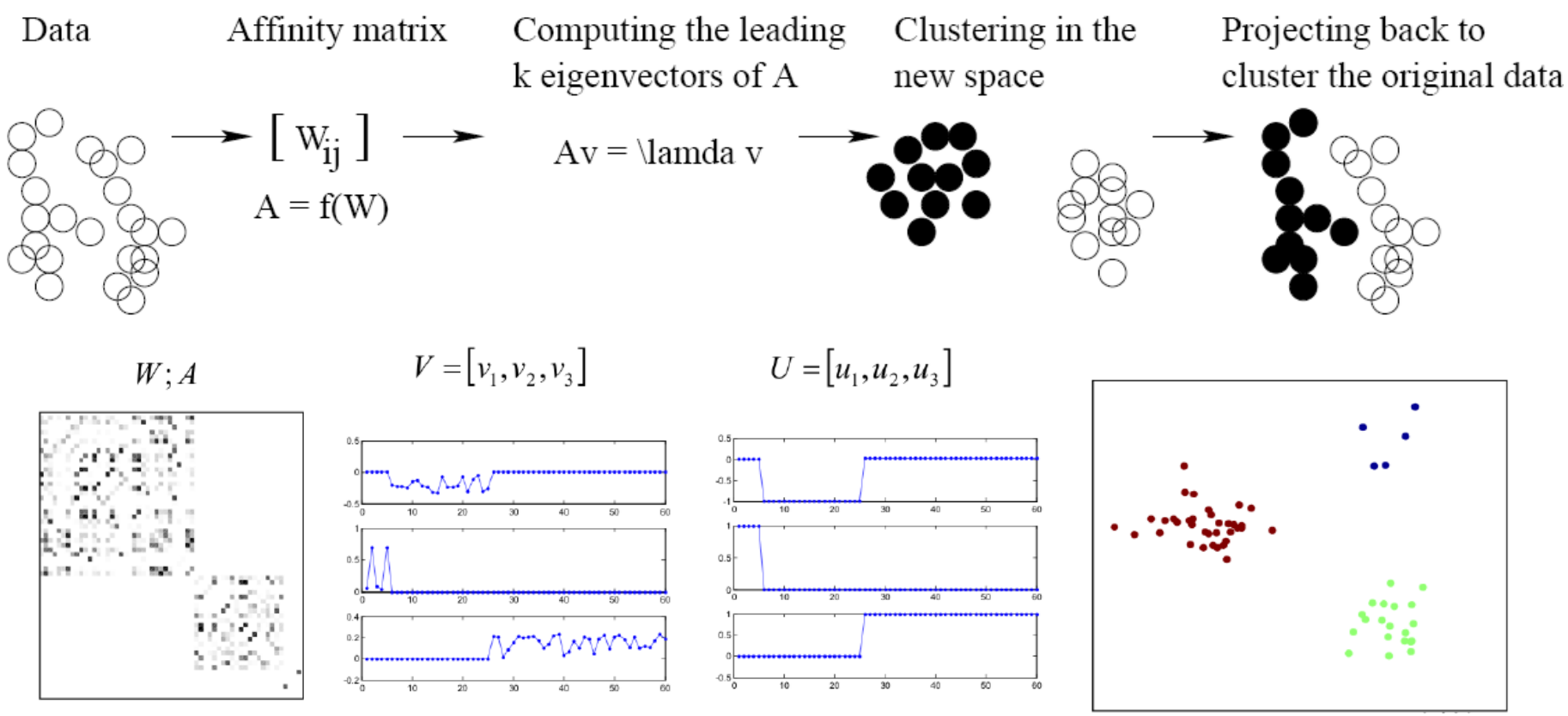
- Derive a matrix A = *f*(W). NJW defines a matrix D to be a diagonal matrix s.t. D_{ii} is the sum of the i-th row of W, i.e.,

$$W_{ij} = e^{-\frac{dist(o_i, o_j)}{\sigma^2}}$$

Then, A is set to $A = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$

- A spectral clustering method finds the k leading eigenvectors of A
  - A vector v is an eigenvector of matrix A if Av = λv, where λ is the corresponding eigen-value
- Using the k leading eigenvectors, project the original data into the new space defined by the k leading eigenvectors, and run a clustering algorithm, such as *k*-means, to find k clusters
- Assign the original data points to clusters according to how the transformed points are assigned in the clusters obtained

Spectral Clustering: Illustration and Comments



- Spectral clustering: Effective in tasks like image processing
- Scalability challenge: Computing eigenvectors on a large matrix is costly
- Can be combined with other clustering methods, such as bi-clustering

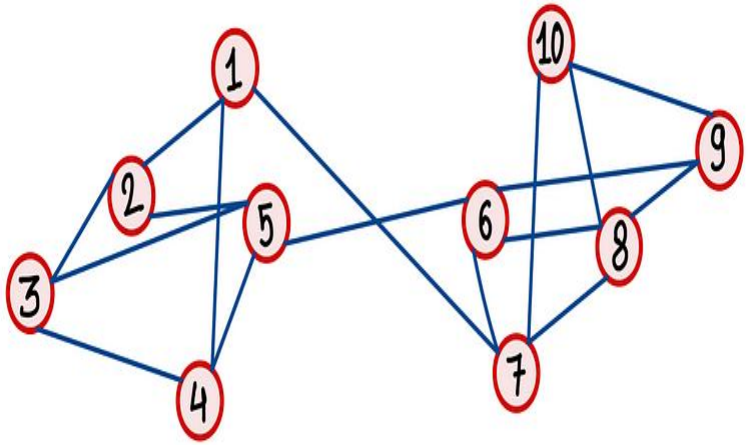
- Spectral clustering is one of the most popular forms of multivariate statistical analysis
- Spectral Clustering uses the connectivity approach to clustering', wherein communities of nodes (i.e. data points) that are connected or immediately next to each other are identified in a graph.
- The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters.
- Spectral Clustering uses information from the eigenvalues (spectrum) of special matrices (i.e. Affinity Matrix, Degree Matrix and Laplacian Matrix) derived from the graph or the data set.
- Spectral clustering methods are attractive, easy to implement, reasonably fast especially for sparse data sets up to several thousand. Spectral clustering treats the data clustering as a graph partitioning problem without making any assumption on the form of the data clusters.

Difference between Spectral Clustering and Conventional Clustering Techniques

- Spectral clustering is flexible and allows us to cluster non-graphical data as well.
- It makes no assumptions about the form of the clusters.
- Clustering techniques, like K-Means, assume that the points assigned to a cluster are spherical about the cluster centre. This is a strong assumption and may not always be relevant.
- In such cases, Spectral Clustering helps create more accurate clusters. It can correctly cluster observations that actually belong to the same cluster, but are farther off than observations in other clusters, due to dimension reduction.
- The data points in Spectral Clustering should be connected, but may not necessarily have convex boundaries, as opposed to the conventional clustering techniques, where clustering is based on the compactness of data points.
- Although, it is computationally expensive for large datasets, since eigenvalues and eigenvectors need to be computed and clustering is performed on these vectors.
- Also,for large datasets, the complexity increases and accuracy decreases significantly.

Spectral Clustering Matrix Representation Adjacency and Affinity Matrix (A)

- The graph (or set of data points) can be represented as an Adjacency Matrix, where the row and column indices represent the nodes, and the entries represent the absence or presence of an edge between the nodes (i.e. if the entry in row 0 and column 1 is 1, it would indicate that node 0 is connected to node 1).



- An Affinity Matrix is like an Adjacency Matrix, except the value for a pair of points expresses how similar those points are to each other. If pairs of points are very dissimilar then the affinity should be 0. If the points are identical, then the affinity might be 1. In this way, the affinity acts like the weights for the edges on our graph.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Laplacian Matrix (L)

Degree Matrix (D)
A Degree Matrix is a diagonal matrix, where the degree of a node (i.e. values) of the diagonal is given by the number of edges connected to it. We can also obtain the degree of the nodes by taking the sum of each row in the adjacency matrix.

$$\mathrm{D} = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

- **Laplacian Matrix (L)**
- This is another representation of the graph/data points, which attributes to the beautiful properties leveraged by Spectral Clustering. One such representation is obtained by subtracting the Adjacency Matrix from the Degree Matrix (i.e. L = D – A).

$$L = \begin{bmatrix} 3 & -1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 3 \end{bmatrix}$$

- **Spectral Gap**: The first non-zero eigenvalue is called the Spectral Gap. The Spectral Gap gives us some notion of the density of the graph.
- **Fiedler Value:** The second eigenvalue is called the Fiedler Value, and the corresponding vector is the **Fiedler vector**. Each value in the Fiedler vector gives us information as to which side of the decision boundary a particular node belongs to.
- Using L, we find the first large gap between eigenvalues which generally indicates that the number of eigenvalues before this gap is equal to the number of clusters.
- What is Spectral Clustering and how its work? (mygreatlearning.com)